Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Big data y Deep Learning para la detección de objetos en imágenes

TITULACIÓN: Grado en Ingeniería Telemática

AUTOR: Alessandro Iván Fava Fernández

DIRECTORA: Esther Salamí San Juan

FECHA: 31 de Octubre de 2017

Título: Big data y Deep Learning para la detección de objetos en imágenes

Autor: Alessandro Iván Fava Fernández

Director: Esther Salamí San Juan

Fecha: 31 de Octubre de 2017

Resumen

En este proyecto podemos ver dos campos que actualmente están en auge, Deep Learning y Big Data. Concretamente la detección de objetos que corresponde a la visión artificial, no se trata simplemente de la detección de objetos dentro de una imagen sino que se busca solucionar un problema.

El problema a solucionar es cuando no poseemos un dataset de imágenes lo suficientemente grande como para poder realizar una detección óptima. En la actualidad este problema es recurrente, si desarrollamos una aplicación para buscar una especie concreta de animal, y está en peligro de extinción es muy probable que el número de imágenes que poseemos será demasiado pequeño para conseguir nuestro objetivo.

Para solucionar este problema hemos empleado dos tecnologías. La primera, que ya ha sido probada su eficacia a la hora de detectar rostros. Esta es Haar-Cascade sobre la librería OpenCV.

El segundo método es relativamente nuevo y se denomina transfer learning. En el caso de transfer learning utilizamos el framework desarrollado por Microsoft denominado Cognitive Toolkit y también utilizamos servicios de cloud de Microsoft como Azure.

Para probar la eficacia de los métodos que tratamos en el proyecto hemos probado su capacidad de detección y clasificación con diferentes elementos. Hemos empleado imágenes de relojes, lobos, ovejas y señales de helipuertos.

Este último objeto lo hemos escogido porque no hay un dataset de imágenes de él y esto hace que se acerque más a un caso real.

Finalmente probaremos de forma experimental si todo esto es posible y que resultados e ideas podemos extraer de todo ello de cara al futuro.

Palabras claves:
Deep Learning, Máquina Virtual, dataset, framework, script, descriptors, release.

Title: Big data and Deep Learning for the detection of objects within images

Author: Alessandro Iván Fava Fernández

Director: Esther Salamí San Juan

Date: October 31st 2017

Overview

In this project we can see two fields that are very popular nowadays, Big Data and Deep Learning. In particular, the object detection is in the field of artificial vision. But we are not looking for detect an object within the image, we try to solve a problem.

The problem to resolve is when we don't have a dataset large enough to get an optimal result. These days this problem persist, if we develop an application to looking for a rare animal species that is endangered, is more probably that the images that we have from this animal will be not enough to get a good result for detect them.

To resolve this problem we have used two different technologies. The first one, it has proved it efficiency detecting faces, this is the Haar-Cascade over OpenCV. The second one is relatively newest, it is called transfer learning.

In the case of transfer learning we use a framework called Cognitive Toolkit developed by Microsoft and also we use Azure cloud service developed by Microsoft.

To prove the efficiency of this two methods we have tried to detect different objects: Watches, wolfs, sheep and helipad signs. We have chosen this last object because there isn’t any dataset of it and this makes it closer to a one real case.

Finally, we will test in practical way. If all we planned before is possible we will be able to find out interesting ideas for the future.

Key words:
Deep Learning, Virtual Machine, dataset, framework, script, descriptors, release.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN 1

1.1. Motivación y objetivos 1

1.2. Organización del documento 1

CAPÍTULO 2. MARCO CONCEPTUAL E HISTÓRICO 3

2.1. Marco conceptual 3
2.1.1. Image Processing 4
2.1.2. Machine learning 4
2.1.3. Machine vision 4

2.2. Marco histórico y evolución 4

2.3. ¿Qué es un clasificador? 7
2.3.1. Aprendizaje supervisado 9
2.3.2. Aprendizaje no supervisado 9
2.3.3. Aprendizaje semi-supervisado 10

CAPÍTULO 3. DETECCIÓN DE OBJETOS CON HAAR-CASCADE 12

3.1. ¿Qué es OpenCV? 12

3.2. Funcionamiento de Haar-cascade 13
3.2.1. Imagen integral 14
3.2.2. Adaboost (Adaptative Boost) 14
3.2.3. Cascada de clasificadores 20

CAPÍTULO 4. FASE EXPERIMENTAL I: HAAR-CASCADE 22

4.1. Instalación entorno experimental 22

4.2. Desarrollo experimental 22
4.2.1. Obtención de las imágenes 22
4.2.2. Creación de imágenes positivas 25
4.2.3. Entrenamiento del clasificador 27
4.2.4. Test de precisión del clasificador 29

4.2. Conclusiones 32

CAPÍTULO 5. DETECCIÓN DE OBJETOS CON TRANSFER LEARNING 33

5.1. Deep learning 33

5.2. ¿Qué es CNTK (Cognitive Toolkit)? 34

5.3. Convolutional Neural Networks (CNN ó ConvNets) 36
5.3.1. Convolución 37
5.3.2. Funciones de activación (*Activation functions*) 40

5.3.3. Agrupación (*Pooling*) 41
5.3.4. Capa totalmente conectada (*fully connected layer*) 43

5.4. Transfer learning 44
5.4.1. Transferencia de conocimientos en DCNN 45

CAPÍTULO 6. FASE EXPERIMENTAL II: TRANSFER LEARNING 47

6.1. Entorno experimental 47
6.1.1. Azure 47
6.1.2. Deep Learning Virtual Machine (DLVM) 48
6.1.3. Despliegue de DLVM 48

6.2. Desarrollo experimental 49
6.2.1. Obtención de las imágenes 49
6.2.2. Entranamiento y evaluación 51

6.3. Conclusiones 54

CONCLUSIONES 56

BIBLIOGRAFÍA 58

ANEXOS 1

ANEXO I. MULTILAYER PERCEPTRON (MPL) 1

ANEXO II. INSTALACIÓN OPENCV 3

ANEXO III. HAAR-CASCADE SCRIPTS 5

ANEXO IV. EVALUACIÓN DEL RENDIMIENTO DE UN CLASIFICADOR 7

ANEXO V. TRANSFER LEARNING SCRIPTS 8

CAPÍTULO 1. INTRODUCCIÓN

1.1. Motivación y objetivos

En la actualidad hay un creciente uso de tecnologías que emplean Big Data y Deep learning a la hora de encontrar algo, es decir, tanto a nivel lingüístico como en imágenes se trata de encontrar palabras claves u objetos en imágenes para un fin. Este fin pueden ser recomendaciones, la ubicación de un objeto en una tienda, tareas de rescate como encontrar una persona, etc.

Por ello este proyecto se centra en la detección de objetos y la clasificación de imágenes. No se trata simplemente en diferenciar los elementos que conforman una imagen, en la actualidad hay decenas de aplicaciones que nos permiten esto, el objetivo va más allá intentando buscar la solución a un problema.

Este problema es recurrente cuando hablamos de Big Data y Deep Learning y es la falta de muestras. Comúnmente los algoritmos de clasificación y detección de objetos requieren de un gran número (decenas de miles) de muestras para entrenar el sistema. Pero no siempre disponemos de esta cantidad entonces nos planteamos: ¿Es posible alguna alternativa u obtener un resultado óptimo?

Durante la realización de este proyecto intentaremos ver como resuelven este problema algunas de las tecnologías existentes. Las tecnologías que emplearemos son dos. Una más reciente denominada *transfer learning* y otra no tan reciente pero con resultados óptimos en otros campos llamada *Haar-Cascade*.

El principal objetivo es comprobar la eficacia y viabilidad de estas tecnologías, pero para ello emplearemos diversos tipos de muestras. Algunas sencillas como detectar un reloj en una imagen y otras más complejas, tanto como para la detección como para obtener imágenes, que es caso de una señal de helipuerto.

Este segundo objeto es escogido para darle un sentido práctico y acercarnos a un caso real, no como objetivo. Ya que no hay dataset de imágenes de este objeto. Y por otro lado si logramos unos resultados satisfactorios, este tipo de objeto suele ser empleado tanto para salvamento como para aterrizaje de vehículos aéreos con conducción autónoma.

1.2. Organización del documento

El documento se encuentra dividido en seis partes, tres partes son teóricas y dos son a nivel experimental y una última que corresponde a las conclusiones:

- La primera parte el capítulo 2: Marco conceptual e histórico, el objetivo es de explicar los diferentes campos que encontramos en la visión artificial y como están relacionados entre ellos. También explicar la evolución que

ha habido a la hora de detectar objetos dentro de las imágenes, tanto las técnicas como las tecnologías utilizadas a lo largo de la historia.

- Capítulo 3: Detección de objetos con Haar-Cascade. En esta segunda parte explicaremos Haar-Cascade como método de detección de objetos, así como su funcionamiento y la librería principal que necesitamos para trabajar con el que es OpenCV.

- Seguidamente explicamos en el capítulo 4 el funcionamiento a nivel experimental de Haar-Cascade. Explicamos tanto la preparación del entorno de trabajo como el desarrollo con los recursos (imágenes) utilizados y unas conclusiones respecto a los resultados obtenidos

- La tercera parte corresponde al capítulo 5: Detección de objetos con transfer learning. En este capítulo explicamos que es transfer learning, que herramientas empleamos, como el framework CNTK (Cognitive Toolkit) desarrollada por Microsoft. También explicaremos el funcionamiento del clasificador utilizado que es DCNN (Deep Convolutional Neural Network)

- En el capítulo 6 y segunda fase experimental vemos el funcionamiento de transfer learning a nivel práctico. De la misma forma que en el capítulo anterior, analizaremos la instalación y preparación del entorno. También analizaremos el desarrollo con las diferentes herramientas utilizadas, y finalmente unas conclusiones para analizar brevemente los resultados obtenidos si eran los esperados.

- Esta parte final corresponde a las conclusiones, donde analizamos todas las ideas extraídas de realización del proyecto así como una evaluación de los objetivos planteados inicialmente.

CAPÍTULO 2. MARCO CONCEPTUAL E HISTÓRICO

Actualmente existen miles de aplicaciones que emplean la detección de objetos. Un ejemplo de esto es la detección de rostros en las cámaras de los móviles. ¿Pero sabemos realmente qué hay detrás de todo esto y su evolución a lo largo de los años? Estas son cuestiones que resolveremos en este apartado.

2.1. Marco conceptual

Dentro de una imagen podemos encontrar diversos objetos y formas, las cuales queremos detectar o identificar, pero antes de entrar en más detalle sobre la detección y reconocimiento de objetos debemos conocer sobre qué campo estamos trabajando y cómo ha evolucionado.

Todo este trabajo se desarrolla dentro del campo de la visión artificial. A la hora de definir qué es la visión artificial y su objetivo nos encontramos con dos puntos de vista. Por una parte a nivel biológico se trata de conseguir replicar el sentido de la vista y todas sus capacidades. Por otro lado desde el punto de vista de la ingeniería el objetivo no es simplemente el de replicar, sino que se trata de automatizar tareas que realizamos con la vista, por ejemplo identificar un objeto. Normalmente esta automatización sobrepasa las capacidades humanas.

Ambos objetivos están relacionados, ya que desde siempre en la creación de nuevos sistemas el ser humano se ha fijado en la naturaleza.

La visión artificial implementa técnicas y tecnologías de otros campos, de la misma forma que otros campos utilizan la visión artificial como una herramienta para obtener resultados u objetivos.

En la imagen siguiente podemos ver los campos principales que envuelven la visión artificial. Empleando la **Fig. 2.1** nos centraremos en tres campos: *Image Processing*, *Machine Learning* y *Machine Vision.*

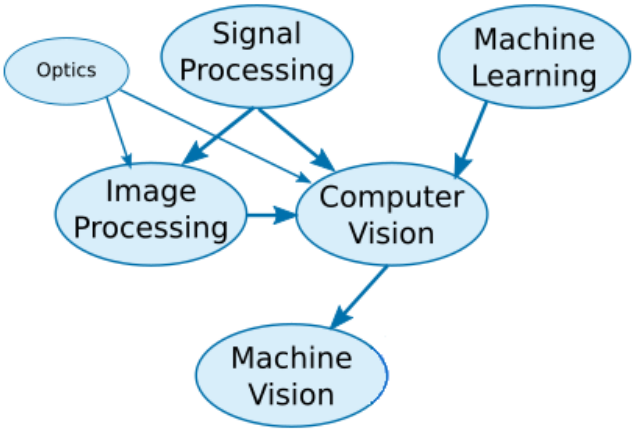

Fig. 2. 1 Diagrama de los diferentes campos relacionados con *computer visión* [6]

2.1.1. Image Processing

El procesado de imágenes dentro de la ingeniería y la computación se refiere a cualquier forma de procesado donde el flujo de entrada es una imagen. Mediante el procesado de una imagen lo que buscamos es mejorar la calidad de la imagen o facilitar la búsqueda de algo dentro de la imagen.

Para ello se aplican diversas técnicas, que pueden ir desde la aplicación de filtros, a capturar imágenes con cámaras con una alta resolución de píxeles y aplicación de contrastes.

2.1.2. Machine learning

El campo del aprendizaje automático se encuentra en auge actualmente. La finalidad de este es conseguir que una máquina sea capaz de identificar o entender el significado de un ítem (ítem hace referencia a objeto, imagen, frase, etc.) que no haya visto antes.

Para llegar a este objetivo la máquina es entrenada con ejemplos, miles de ellos. Hay dos campos dentro de la inteligencia artificial que son fundamentales: Uno es el reconocimiento de patrones, donde se identifican rasgos comunes en los ejemplos, y otro, es la teoría computacional del aprendizaje, donde se buscan los algoritmos más eficientes para el aprendizaje, es decir, como mejora su capacidad de detección.

2.1.3. Machine vision

Este tercer campo es más bien una diferencia conceptual a la hora de englobar el uso de técnicas de visión artificial, ya que la principal diferencia es que se denomina *machine vision* a todos los procesos donde después del análisis visual esto conlleve una acción posterior inmediata o no, es decir, engloba el análisis de datos de un sistema de vigilancia hasta el análisis del entorno en un robot para su estabilidad.

En el caso de un sistema de vigilancia a partir de lo que se captura por las cámaras no se realiza ninguna acción. Por otra parte, la cámara de un robot analiza el terreno para que no se desestabilice, realizando acciones inmediatas.

2.2. Marco histórico y evolución

En el apartado anterior hemos podido ver las definiciones de los campos que engloban el proyecto. Ahora vamos a ver la evolución de la visión artificial y cómo las técnicas empleadas para la detección de objetos han cambiado.

En este apartado intentaremos explicar su evolución, pero no se puede establecer una línea cronológica de los sucesos, ya que muchos estudios se realizaron en paralelo en la misma época, pero sí podemos ver su efecto.

El primer problema que surgió a la hora de detectar un objeto era como hacer invariante el objeto al punto de visión del objetivo de la cámara. Para ello el objeto debía estar alineado con el otro objeto al cual es semblante.

Con alineado se refiere a que se realiza la transformada de la imagen con el objeto mediante algún modelo de transformación, a partir de las transformada se calcula la diferencia entre los puntos claves de las dos imágenes y se realiza la detección, en la **Fig. 2.2** podemos ver un ejemplo. Todos estos avances y mejoras sucedieron durante el periodo comprendido entre 1972 y 1993.

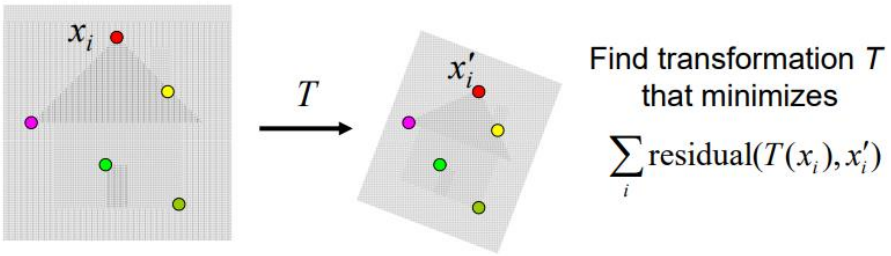

Fig. 2. 2 Ejemplo de transformada mediante modelos [1]

A la vez que se estudiaba cómo identificar imágenes mediante transformadas también se buscaba de realizarlo de una forma computacionalmente más eficiente.

Esta forma era mediante el desglose del objeto a detectar en formas conocidas, en este caso se emplearon figuras geométricas.

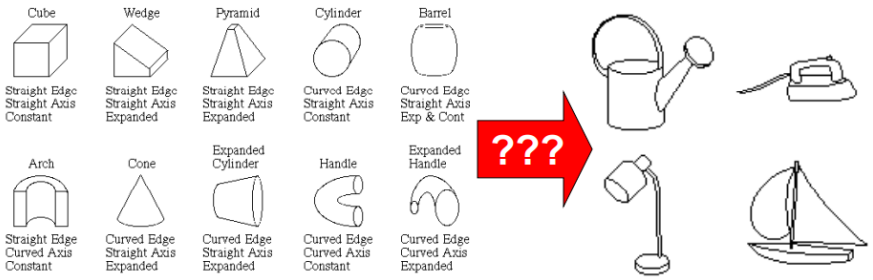

Fig. 2. 3 Descomposición de objetos en formes geométricas conocidas [1]

Después de intentar separar el objeto en componentes y utilizar elementos geométricos para describir la imagen, se planteó usar técnicas basadas en la apariencia, como los histogramas para mejorar la detección. Todos estos avances y estudios se realizaron entre 1991 y 1995.

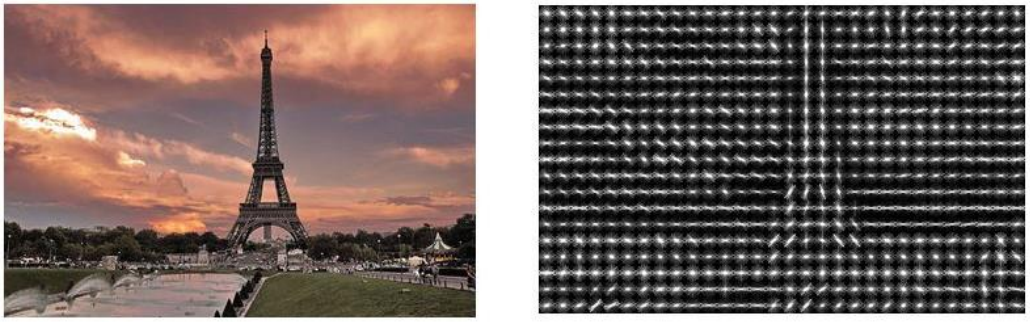

Fig. 2. 4 Histograma de la Torre Eiffel [4]

En paralelo con estos estudios de cómo describir los componentes de la imagen, también se está buscando una forma más rápida y focalizada de detectar los objetos dentro la imagen.

No hay un periodo exacto de este avance pero el primer artículo respecto a la aproximación por ventana deslizante fue en 1991. El objetivo de esta técnica es establecer una ventana de un alto y ancho determinado que irá recorriendo toda la imagen y a la vez analizando si detecta el objeto por donde pasa.

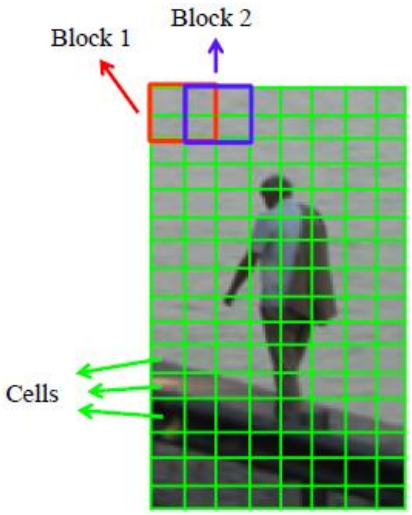

Fig. 2. 5 Ejemplo ventana deslizante [16]

Finalmente, a principios de 2000 se utilizaban todas las técnicas conocidas, combinándolas de la forma más óptima posible. Pero actualmente a la hora de detectar objetos se pueden emplear dos técnicas bastante diferentes.

Por un lado nos encontramos con el uso de descriptores y puntos claves del objeto a detectar, de manera que podamos identificar ese objeto concretamente e inequívocamente en cualquier imagen.

Por otro lado nos encontramos con el uso de clasificadores, aunque también emplean descriptores no lo hacen de la misma manera. Mediante los clasificadores lo que realizamos es la búsqueda de una etiqueta creada a partir de una gran cantidad de muestras de un objeto, por ejemplo, tenemos muestras

de una gran variedad de tipos de relojes y podemos identificar cualquier reloj, no buscamos uno concreto.

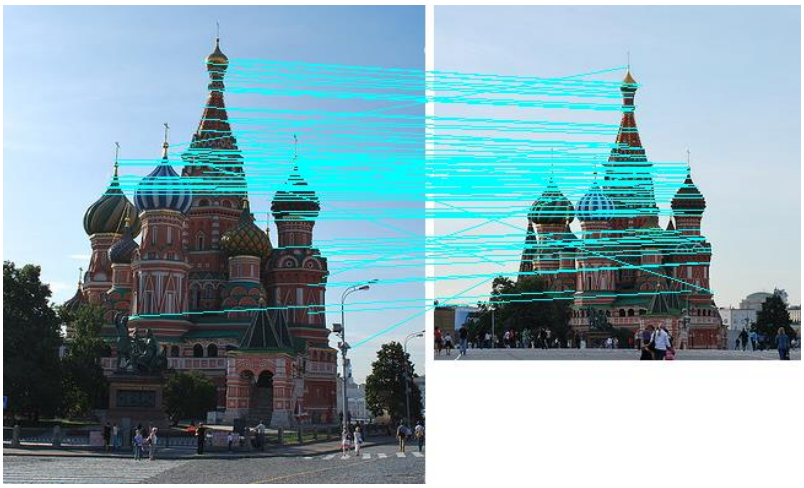

Fig. 2. 6 Detección de objeto mediante *keypoints* y descriptores

Fig. 2. 7 Detección de rostros mediante un gran número de muestras de este [5]

2.3. ¿Qué es un clasificador?

Un clasificador es un algoritmo que implementa una forma de clasificar. Esta definición hace referencia a la clasificación estadística, donde a partir una entrada de información debemos obtener como salida la categorización de ella.

Si hablamos del funcionamiento de un clasificador tiene otros factores que afectan a la categorización o clasificación de la información.

En este proyecto tratamos la clasificación de imágenes, pero también intervienen otros procesos antes de etiquetar las imágenes de la **Fig. 2.8** podemos ver cómo funciona el sistema de clasificación.

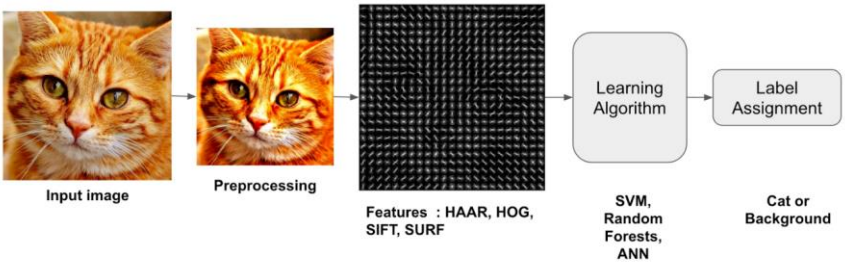

Fig. 2. 8 Proceso de etiquetado de una imagen [3]

El proceso inicia con una imagen como entrada la cual deberemos clasificar, pero antes esta imagen se ha de pre-procesar, normalmente definiendo un determinado tamaño y/o transformando la imagen a escala de grises.

Una vez la imagen es apta se aplicará un descriptor (ejemplos de descriptores son: HAAR, HOG, SIFT, SURF, etc.) que nos permitirá obtener una gran variedad de características de la imagen. A partir de estas características empleamos un clasificador que va a etiquetar las imágenes según sus algoritmo y las características que le resulten útil de la imagen.

No nos centraremos en el funcionamiento de los descriptores ni en explicar la diferencia entre ellos, ya que existe una gran variedad de estos y es un tema muy extenso.

Actualmente existen una gran variedad de clasificadores según los métodos y algoritmos que emplean. En la **Fig. 2.9** podemos un diagrama que clasifica la mayoría de ellos**.**

Fig. 2. 9 Diagrama de clasificadores [11]

Una vez hemos visto cómo se etiqueta una imagen, la pregunta que surge es: ¿Cómo sabe nuestro clasificador qué etiqueta ponerle a la imagen de entrada? La respuesta depende del tipo de aprendizaje que utilicemos:

2.3.1. Aprendizaje supervisado

En el aprendizaje supervisado o dirigido lo que se hace previamente de clasificar una imagen desconocida para el clasificador es prepararlo. Para preparar al clasificador lo que hacemos es entrenarlo con una gran cantidad de imágenes (*dataset*) etiquetadas. Estas imágenes contienen el objeto que queremos detectar, de esta forma estamos enseñando a nuestro clasificador a identificar concretamente este objeto.

Mediante este entrenamiento previo nuestro clasificador ha creado un modelo que es capaz de mapear nuevos ejemplos e identificar al objeto que tiene como objetivo.

Para obtener unos resultados óptimos empleando este método de aprendizaje necesitaremos de una gran cantidad de muestras para entrenar y otra gran cantidad de muestras para testear nuestro clasificador.

Claramente la contraparte de esta forma de aprendizaje es lo costoso que resulta obtener un *dataset* de imágenes etiquetadas, ya que en muchos casos dependiendo de nuestro objetivo puede resultar algo utópico.

Algunos de los clasificadores empleados para este método son kNN (*k - Nearest Neighbour*), árboles de decisión o vectoriales como SVM (*Support Vector Machine*)

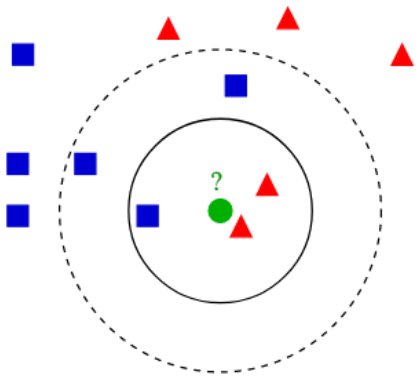

Fig. 2. 10 Ejemplo funcionamiento kNN [12]

2.3.2. Aprendizaje no supervisado

En el aprendizaje no supervisado partimos de un dataset no etiquetado, por lo que no sabemos que podemos encontramos o cual es la solución que debería darnos el clasificador. Debido a ello el clasificador se basa en las características obtenidas por el descriptor para clasificar las entradas.

El objetivo de este tipo de aprendizaje es obtener mayor información sobre los datos tratados, intentando modelar la estructura o distribución subyacente.

Para modelar esta estructura se busca una asociación entre ellos, mediante probabilidades condicionales, o se agrupan (*Clustering*) según las características obtenidas por el descriptor.

La agrupación (*Clustering)* se emplea para obtener grupos de datos relacionados dentro del dataset, como por ejemplo los clientes que invierten en el mismo ámbito comercial.

Por otro lado, la asociación se emplea para descubrir reglas dentro de los datos, como por el ejemplo los clientes que compran cierto producto compran este también.

Algunos clasificadores empleados pueden ser neuronales como *EM-algorithm* o de tipo agrupador como *K-means*.

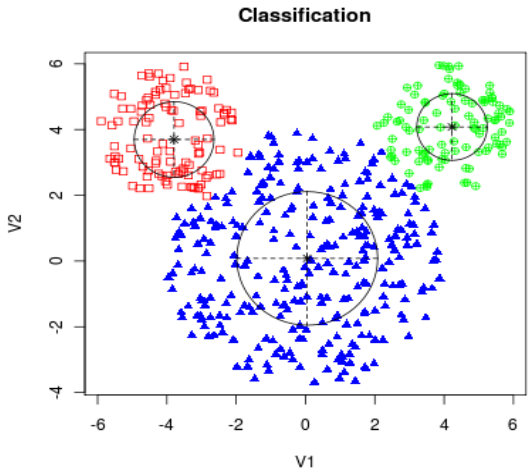

Fig. 2. 11 Ejemplo clasificador k-means [12]

2.3.3. Aprendizaje semi-supervisado

El aprendizaje semi-supervisado ha tenido muchos avances en estos últimos años, ya que nos permite reducir el gran coste que supone obtener un dataset de imágenes etiquetadas.

Mediante este nuevo método empleamos una pequeña cantidad de muestras con imágenes etiquetadas que no suponen un gran coste y un gran dataset de imágenes sin etiquetar.

Los estudios recientes demuestran que el emplear aunque sea un número muy pequeño de imágenes etiquetadas nos permite mejorar considerablemente la eficacia del entrenamiento y por tanto la clasificación.

Para esta forma de aprendizaje se suelen utilizar algoritmos de agrupación, que se emplean en aprendizaje no supervisado, ya que mediante los datos etiquetados podemos ver qué conjunto de muestras no etiquetadas son relacionadas con este grupo, y por tanto obtener un mayor número de muestras correctas.

Se puede suponer que no todas las muestras asociadas al conjunto de muestras etiquetadas pertenezcan a este conjunto pero aun siendo así el número de muestras etiquetadas aumentará.

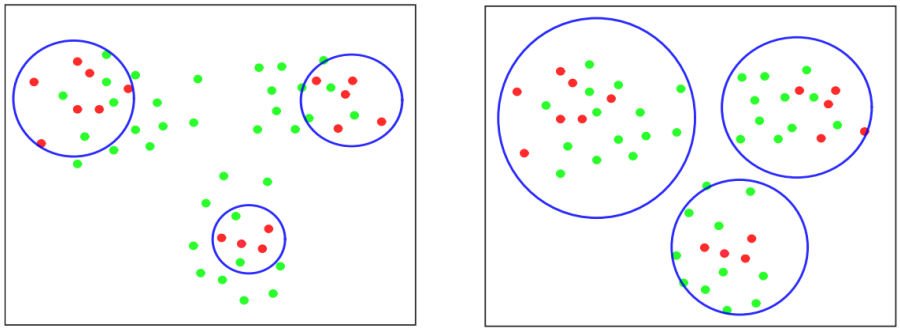

Fig. 2. 12 Agrupación de muestras no etiquetadas con muestras etiquetadas [14]

CAPÍTULO 3. DETECCIÓN DE OBJETOS CON HAAR-CASCADE

Existen diversos métodos para detectar objetos, actualmente uno de los más empleados para detección de rostros y objetos es Haar-Cascade, por ello es importante antes de empezar directamente con la práctica entender cómo funciona esta solución y el porqué de esta elección.

3.1. ¿Qué es OpenCV?

Opencv (*OpenSource Computer Vision*) es una librería de código abierto de visión artificial y *machine learning.*

Esta librería fue desarrollada originalmente por *Intel* lanzando su primera *alfa* en Enero de 1999 y su primera versión estable en 2006. Tres años después en Octubre de 2009 se lanzó su segundo *release OpenCV v2.*

La librería tiene una licencia BSD, que permite que el usuario pueda usar y modificar el código. Por otra parte este tipo de licencia permite que pueda ser empleada tanto para propósitos comerciales como para la investigación.

En la actualidad muchas grandes multinacionales la emplean, como Sony, Google, Yahoo, Honda, etc. De la misma manera cuenta con una comunidad de más de 47000 persona y con más de 7 millones de descargas.

Originalmente fue escrita para trabajar con lenguaje C/C++, aunque su mayor virtud es que es multiplataforma. Actualmente podemos utilizarla con otros lenguajes como Java, Objective C, Python y C#. Como hemos comentado se trata de un librería multiplataforma, por lo que podemos usarla en Linux, Windows, Mac OS X, Android e iOS.

Esta librería ha sido muy robusta desde los inicios, por lo que hace 2 años en 2015 lanzó su tercera release (*OpenCV v3*), la cual nos permite trabajar con versiones más recientes de algunos lenguajes, como *Python 3.0.*

En la actualidad OpenCV consta con más de 2500 algoritmos tanto de *machine learning* como de visión artificial. Estos algoritmos permiten identificar objetos, caras, clasificar acciones humanas en vídeo, hacer tracking de movimientos de objetos, extraer modelos 3D, encontrar imágenes similares, eliminar ojos rojos, seguir el movimiento de los ojos, reconocer escenarios, etc

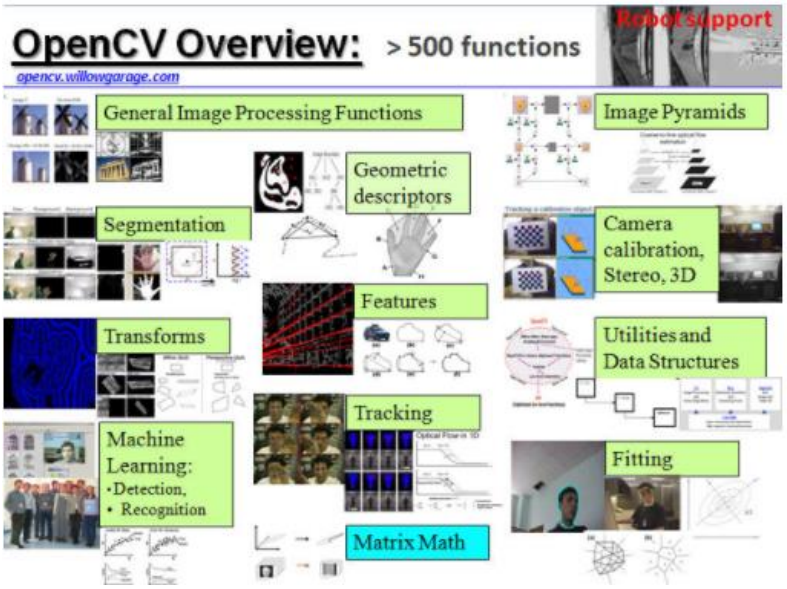

Fig. 3. 1 Funcionalidades de OpenCV [17]

3.2. Funcionamiento de Haar-cascade

En este apartado definiremos el funcionamiento del primer método. El cual emplearemos para la detección de objetos con un número muy reducido de muestras etiquetadas.

Haar-cascade classifier es un método muy efectivo de detección de objetos propuesto por *Paul Viola* y *Michael Jones.* Este método utiliza un descriptor *Haar*, el cual utilizando unos filtros conseguimos una gran cantidad de características de la imagen.

Fig. 3. 2 En esta imagen podemos ver un ejemplo de filtros de Haar [24]

Estos filtros recorren la imagen completa a diferentes escalas y en múltiples posiciones de la imagen, estas múltiples posiciones incluyen el aplicar los filtros con una inclinación de 45º. Las características que se obtienen es la suma de píxeles negros menos la suma de píxeles blancos.
Aplicando estos filtros obtenemos una cantidad muy grande de características de Haar, ya que por cada recorrido que realice un filtro en un tamaño y posición determinada por la imagen obtenemos una característica.

El principal problema que surge al utilizar este descriptor es la cantidad de características que genera y lo costoso que resulta el tiempo de cálculo, así como

el aprendizaje. Para solucionar este problema y aumentar la eficiencia tenemos diferentes técnicas y herramientas.

3.2.1. Imagen integral

Una de las herramientas que utiliza es la imagen integral de la imagen. La imagen integral es una transformación de la imagen original, que nos da como resultado una imagen del mismo tamaño que la original. Pero donde el valor de cada píxel corresponde a la suma de todos los píxel situados arriba a la izquierda en la imagen original.

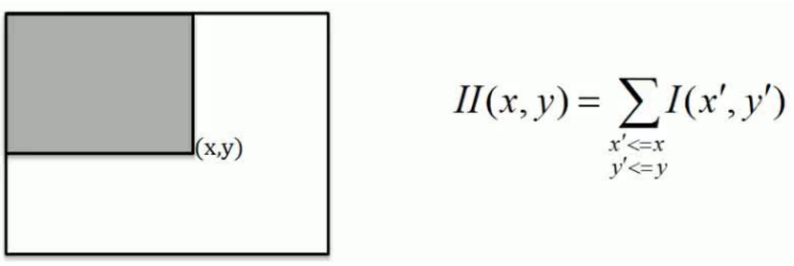

Fig. 3. 3 Explicación visual de imagen integral [24]

Esto nos facilita y agiliza el cálculo de intensidades de cualquier rectángulo dentro de la imagen, y como hemos comentado anteriormente la obtención de las características de haar es la aplicación de los filtros, los cuales realizan una suma de intensidades.

3.2.2. Adaboost (Adaptative Boost)

Una vez obtenidas todas las características de la imagen deberemos aplicar un clasificador adecuado que pueda tratar con todas estas características, este clasificador es Adaboost. Esta herramienta nos permitirá agilizar el proceso de clasificación.

Adaboost emplea un conjunto de clasificadores denominados *decision stump.* Estos clasificadores son arboles de decisión binaria de profundidad uno, y su rendimiento es ligeramente mayor al de uno de clasificación aleatoria. Este tipo de clasificador establece las decisiones a partir de un umbral y separando en dos las muestras, -1 y +1.

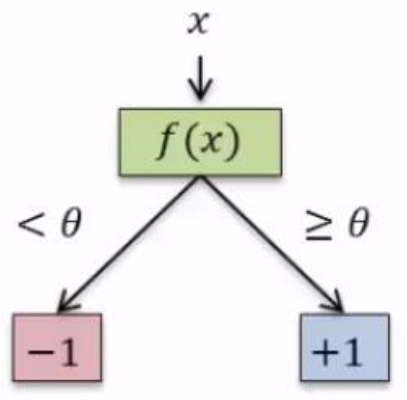

Fig. 3. 4 Diagrama de un *decision stump* [28]

La función en la que se basará el clasificador para decidir será una característica de *Haar,* pero como hemos dicho emplea un conjunto de clasificadores, de manera que combinando este tipo de clasificadores obtenemos una clasificación muy buena.

Para entender mejor el funcionamiento del clasificador *adaboost* cuando lo entrenamos, partamos de un ejemplo:

- Primero contaremos con un número de muestras al que le aplicaremos un primer clasificador.

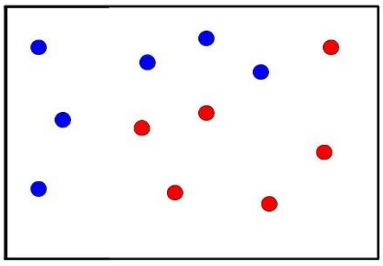

Fig. 3. 5 Diagrama de muestras aleatorias

- Una vez hemos aplicado el primer clasificador obtenemos la siguiente clasificación:

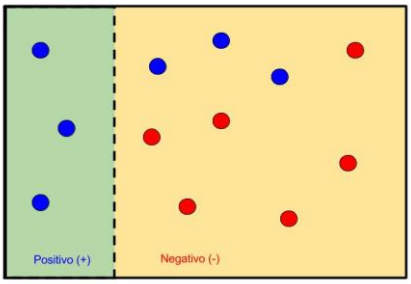

Fig. 3. 6 Utilización del primer clasificador

- *Adaboost* aplica un peso mayor a las muestras clasificadas de forma incorrecta y un peso menor a las muestras clasificadas correctamente.

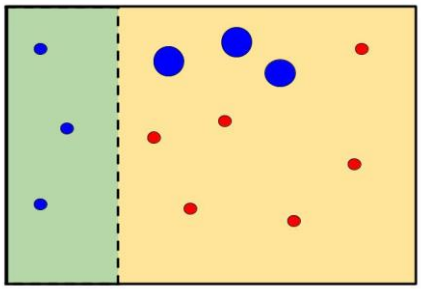

Fig. 3. 7 Ejemplo de cómo cambian los pesos en las muestras

- De esta forma el clasificador aprende de cada clasificador que crea y emplea, al aplicar un siguiente clasificador volverán a cambiar los pesos, ya que las muestras erróneas serán otras y obtendremos lo siguiente:

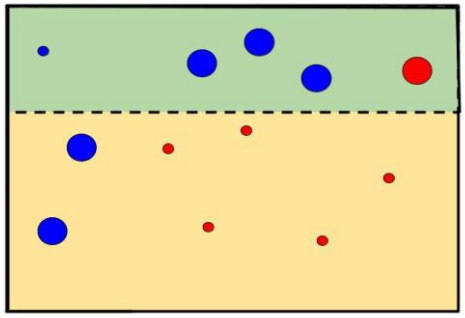

Fig. 3. 8 Utilización del segundo clasificador

- Ahora aplicaremos un tercer clasificador que volverá a ajustar los pesos

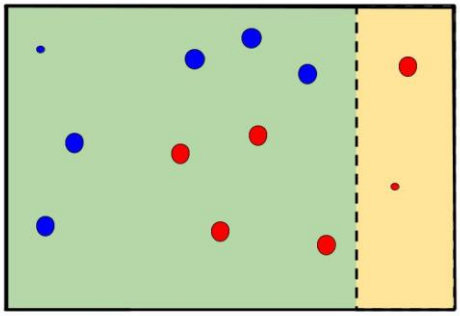

Fig. 3. 9 Utilización tercer clasificador y variación de los pesos

- Ahora se obtendrá la combinación de todos los clasificadores. En cada área veremos qué tipo de clasificación se ha aplicado, es decir si la ha

clasificado mayoritariamente como negativa o positiva, será una región positiva o negativa.

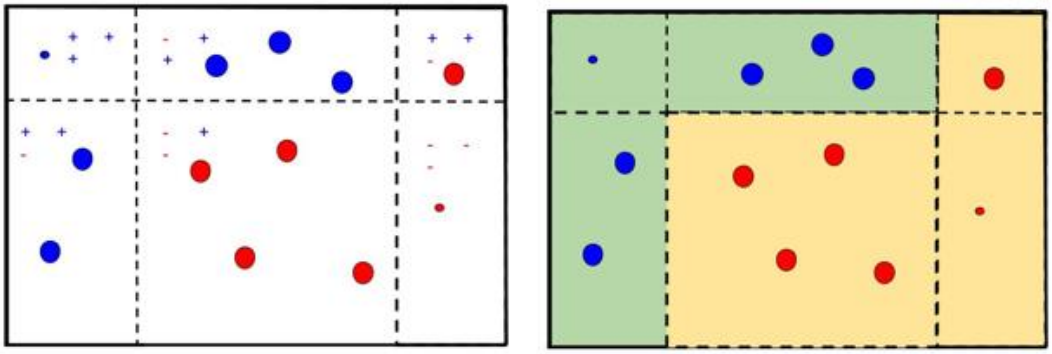

Fig. 3. 10 En esta ilustración vemos cómo se comparan las diferentes clasificaciones y la clasificación resultante

Respecto al clasificador *decision stump* hemos de tener en cuenta un factor. Este factor es cómo decide qué lado del umbral corresponde al negativo o al positivo. Para ello en la formulación del clasificador se aplica un parámetro (alfa) que nos permite discernir en qué situación nos encontramos.

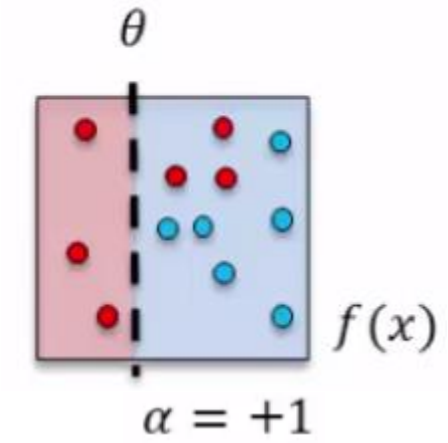

Fig. 3. 11 En esta figura se indica que la zona izquierda corresponde a la zona negativa [28]

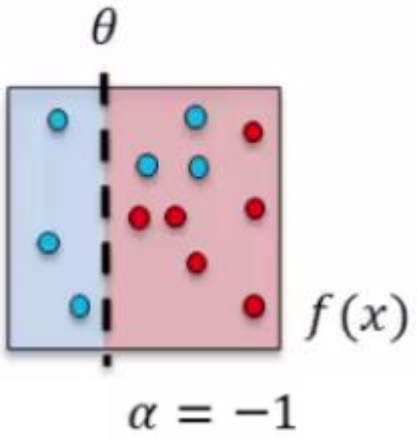

Fig. 3. 12 En esta figura se indica que la zona izquierda corresponde a la zona positiva [28]

$$h(x) = \begin{cases} -\alpha & f(x) < \theta \\ \alpha & f(x) \geq \theta \end{cases}$$

Fig. 3. 13 Formula del *decision stump* (alfa solo puede valer -1 o 1)

De esta forma la eficacia de decisión del clasificador depende únicamente del parámetro alfa y del valor del umbral.

Una vez hemos visto cómo entrenamos el clasificador, el algoritmo de clasificación ha de decidir cuál es el umbral con menor error. Para ello primero deberemos ordenar las muestras y recorrer la función de *f(x).*

Tabla 3. 1 Muestra las ponderaciones y los valores de las muestras

Ponderación	0.5	0.5	0.5	2	1	2	1	1	0.5	0.5	0.5	2
f(x)	48	44	38	35	32	26	22	17	16	12	11	8
Muestras												

Por ejemplo si establecemos el umbral en 48 estamos diciendo que a la derecha del umbral se encuentran las muestras positivas y a la izquierda las negativas, de forma que se vería la siguiente tabla:

Tabla 3. 2 Muestra el criterio de Umbral según el valor de muestra seleccionado

Ponderación	0.5	0.5	0.5	2	1	2	1	1	0.5	0.5	0.5	2
f(x)	48	44	38	35	32	26	22	17	16	12	11	8
Muestras												
Positivo		Negativo										

De manera que tendríamos mal clasificadas siete muestras. Para calcular el error de clasificación debemos de sumar las ponderaciones de las muestras mal clasificados y dividirlas por el número de muestras.

$$Error = \frac{\sum Ponderaciones\ muestras\ mal\ clasificadas}{nº\ total\ de\ muestras}$$

$$Error\ (umbral = 48) = \frac{5}{12} = 0.42$$

Fig. 3. 14 Ejemplo de aplicar la fórmula de cálculo de error

De esta forma calculamos el error con todos los umbrales y nos quedamos con aquel que el error sea menor.

Una vez seleccionado el mejor clasificador. Ahora veremos cómo varían los pesos en las diferentes muestras y cómo combinamos los diferentes clasificadores.

Cada uno de los clasificadores se centra en una característica de Haar diferente, también hay que tener en cuenta que *Adaboost* utiliza un método iterativo, de manera que los pesos anteriores afectan en la siguiente iteración.

$$w_i(t+1) = \begin{cases} \dfrac{1}{\epsilon_t}\dfrac{w_i(t)}{2} & h(x_i) \neq y_i \ (Muestra\ mal\ clasificada) \\ \dfrac{1}{1-\epsilon_t}\dfrac{w_i(t)}{2} & h(x_i) = y_i \ (Muestra\ bien\ clasificada) \end{cases}$$

Fig. 3. 15 Fórmula de cálculo de los pesos de las muestras

En la **Fig. 3.15** podemos ver cómo se obtienen los pesos de la siguiente iteración dependiendo del error del clasificador aprendido anterior y el error actual. De esta forma podemos deducir que el error de clasificación tiene que ser superior al de un clasificador aleatorio, como mínimo ligeramente.

$$\varepsilon_t < 0.5\ (error\ clasificador\ aleatorio) \begin{cases} \dfrac{1}{2\epsilon_t} > 1 & (Muestra\ mal\ clasificada) \\ \dfrac{1}{2(1-\epsilon_t)} < 1 & (Muestra\ bien\ clasificada) \end{cases}$$

Fig. 3. 16 Fórmula para determinar si es una muestra mal o bien clasificada

A partir de la **Fig. 3.16** podemos ver como en el caso de una muestra mal clasificada se multiplicará su peso anterior por un factor superior a 1, y en el caso de una muestra bien clasificada se multiplicará por un factor inferior a 1. De esta forma se ajustaran los pesos de las muestras en cada iteración.

Una vez hemos visto cómo se ajustan los pesos de las muestras en cada iteración, ahora veremos cómo este reajuste continuo afecta al peso del clasificador aprendido.

También veremos cómo unimos los clasificadores aprendidos y cómo creamos el clasificador fuerte, que es la unión de todos los *stump classifier*.

Primero como hemos explicado anteriormente *Adaboost* se basa en los pesos para determinar las características más importantes, es decir, cada clasificador se encargará de clasificar a partir de una característica de *Haar* y en

consecuencia de esto tendrá un error total de clasificación que dependerá de esta característica.

Tabla 3. 3 Representación del error según la muestra y el clasificador utilizado

Característica de Haar	Clasificador	Error (ϵ_T)
$\mathbf{f_1(x)}$	$H_1(x)$	$\epsilon_{T1} = 0.33$
$\mathbf{f_2(x)}$	$H_2(x)$	$\epsilon_{T2} = 0.25$
$\mathbf{f_3(x)}$	$H_3(x)$	$\epsilon_{T3} = 0.12$

Observando la **Tabla 3** podemos ver diferentes características de Haar clasificadas y con errores totales diferentes. Esto hace que dentro del clasificador fuerte que conforman el conjunto de clasificadores débiles (*stumps classifier*) estos clasificadores tienen diferente importancia.

$$H(x) = sign\left(\sum_{i=1}^{T} \alpha_i h_i(x)\right) \qquad \alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Fig. 3. 17 Fórmulas empleadas por Adaboost para unir los clasificadores

En la **Fig. 3.17** podemos ver las expresiones matemáticas que emplea *Adaboost* para unir todos los clasificadores débiles. Por un lado *H(x)* nos indica el signo de la muestra clasificada diciéndonos si es positiva o negativa.

La segunda expresión es un factor que se emplea para cuantificar el peso de un clasificador simple dentro de la unión. Este factor depende básicamente del error total del clasificador. De esta forma analizando la **Tabla 3** podemos ver que el clasificador $H_3(x)$ tendrá un peso superior a los otros dos a la hora de clasificar una muestra.

Como conclusión de todo este apartado referente a *Adaboost* podemos ver de qué se trata de un clasificador ideal para tratar con descriptores que nos dan una gran cantidad de características de una imagen.

3.2.3. Cascada de clasificadores

Finalmente, hemos realizado la clasificación de las muestras de entrenamiento las cuales han pasado por un descriptor y un clasificador, pero *Haar cascade* ofrece una herramienta más que mejora la precisión a la hora de clasificar y es empleando una cascada de clasificadores. Eso quiere decir que las imágenes clasificadas positivamente pasaran por otro clasificador *Adaboost* que servirá como un segundo filtro. La imagen que pase por todos los clasificadores de forma positiva será reconocida como una imagen de aquello que queremos detectar.

Estos filtro o etapas serán tantas como nosotros deseemos, pero mientras más capas haya mayor es el gasto computacional y de recursos que necesitaremos.

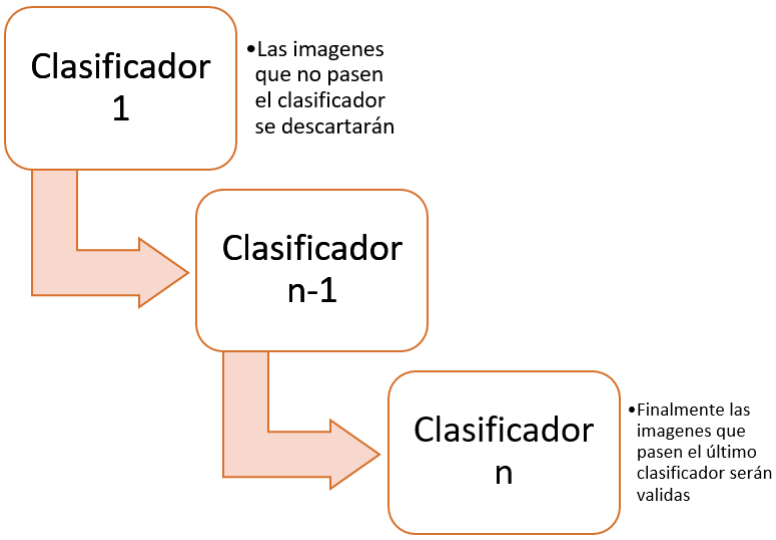

Fig. 3. 18 Diagrama de clasificación en cascada

CAPÍTULO 4. FASE EXPERIMENTAL I: HAAR-CASCADE

En este capítulo explicaremos la parte práctica referente a la utilización de haar-cascade como método de detección de objetos.

Veremos los diferentes pasos, mecanismos y obstáculos que hemos encontrado para obtener resultados, así como las conclusiones acordes al objetivo inicial.

4.1. Instalación entorno experimental

Para poder realizar las pruebas con nuestro propio detector de objetos mediante *haar cascade* ha sido necesario preparar un entorno con las herramientas necesarias.

En este punto mencionaremos los diferentes pasos y elementos que hemos necesitado para realizar las pruebas experimentales.

En primer lugar hemos decidido trabajar sobre el sistema operativo ubuntu, ya que es un entorno muy amigable y donde encontramos mayor cantidad de información respecto a la creación de nuestro detector de objetos. También un elemento determinante en esta decisión es que ubuntu desde la versión 16.04 tiene preinstalado el entorno para trabajar con *python3*, gracias a este entorno podremos ejecutar nuestros *scripts* en *python* sin ningún problema.

Una vez preparado el entorno de trabajo necesitamos instalar *OpenCV*[1]*,* que es nuestra librería principal. OpenCV contiene las funciones necesarias para la detección de objetos, así como para crear nuestro propio clasificador.

La instalación de OpenCV no ha sido sencilla debido a que en el mundo actual existe una facilidad muy grande para obtener información y esto lleva a veces a errores. Con esto nos referimos a que en diferentes lugares podemos encontrar varias formas para realizar una misma acción, y esto provoca que tengamos que escoger y buscar cual es la correcta para nuestro caso particular. Esto implica revisar que versiones tenemos y qué dependencias son necesarias para que todo el entorno funcione.

También hemos empleado *github* por si necesitábamos alguna librería o herramienta para el desarrollo del estudio. Por otro lado, no ha hecho falta instalarla porque ya la utilizabamos en otras asignaturas anteriores.

4.2. Desarrollo experimental

4.2.1. Obtención de las imágenes

1. Anexo II. Instalación OpenCV

En este método se aplica cuando no tenemos suficientes imágenes del objeto a detectar y obtener una detección satisfactoria. Por ello empleamos únicamente una imagen positiva, es decir, sólo utilizamos una imagen del objeto a detectar para entrenar nuestro clasificador.

Pero por otra parte, utilizamos miles de imágenes negativas, es decir, imágenes donde nuestro objetivo a detectar no aparece. Más adelante explicaremos por qué y cómo funciona exactamente esto, pero lo primero es obtener estas imágenes.

Para ello haremos uso de *ImageNet*, que es una de las mayores bases de datos de imágenes del mundo, la cual fue desarrollada por *laboratorio Standford Vision*, *la Universidad de Standford* y *la Universidad de Princeton.*

ImageNet nos permite encontrar imágenes de infinidad de objetos y dentro de estos hay clasificaciones internas, como podemos ver en la **Fig. 5.1** donde encontramos la clasificación de diferentes tipos de relojes.

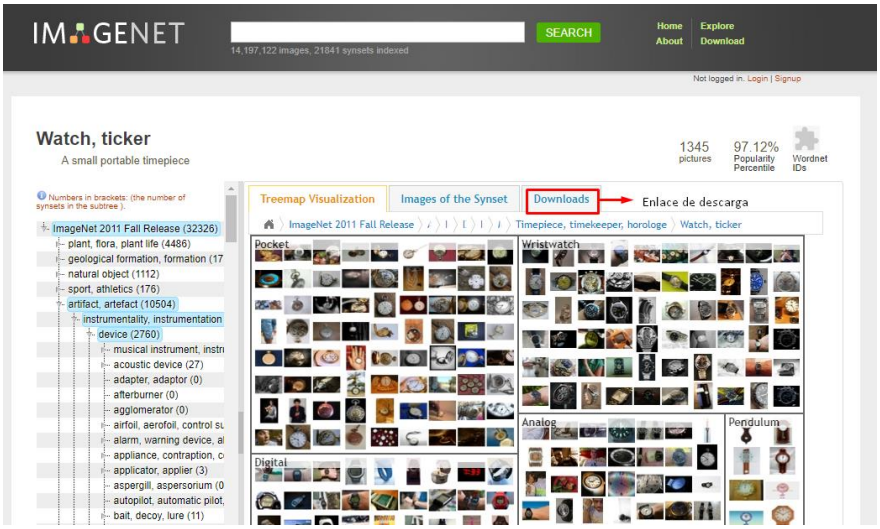

Fig. 4. 1 Ejemplo de data set de imágenes de Relojes en ImageNet [39]

A parte de ofrecernos una gran cantidad de imágenes también nos permite descargarlas, como podemos ver en la pestaña señalada en la **Fig. 4.1**. Esto nos redirecciona a un *synset* con una *url* concreta. Esta *url* contiene las *url* de todas las imágenes de acuerdo a la búsqueda que hemos realizado. En la **Fig. 4.2** podemos ver como es este documento.

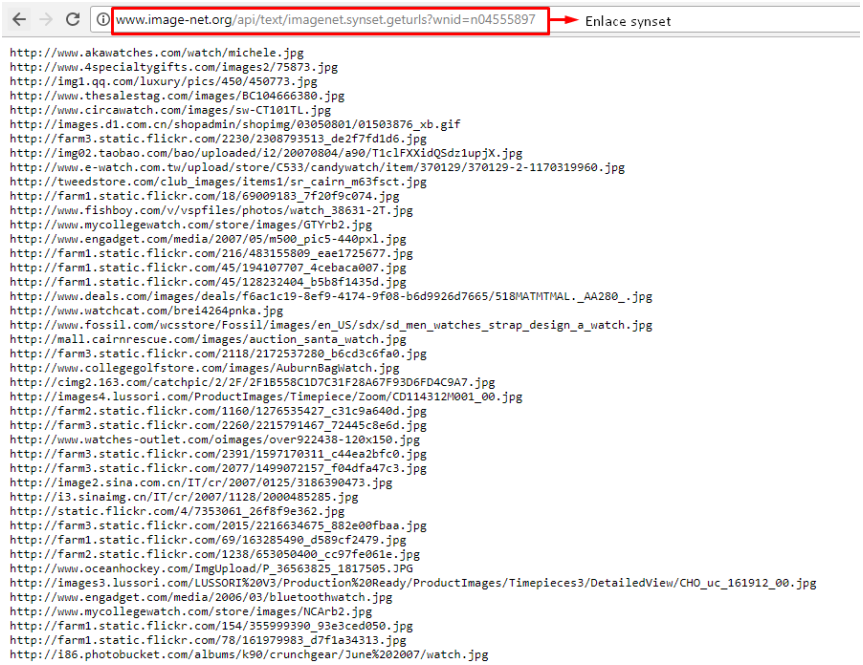

Fig. 4. 2 Ejemplo synset de imágenes de relojes

Empleando este enlace dentro de un *script* en python[1] conseguiremos descargar todas estas imágenes del *synset*, las cuales serán nuestras imágenes negativas.

Para asegurarnos de que estas imágenes sirvan como imágenes negativas realizamos la búsqueda de algo completamente diferente al objetivo.

Tanto en el caso de los relojes como el de las señales de helipuerto empleamos imágenes de personas haciendo deporte. Es posible que en estas imágenes salgan relojes, pero como las imágenes se reducirán a una escala más pequeña (100px x 100px) y se convertirán a escala de grises no deberían de molestar al clasificador.

Una vez obtenidas todas las imágenes negativas debemos de escoger nuestra imagen positiva. En este proyecto se han realizado varias pruebas para obtener diferentes resultados.

Primero, para comprobar el funcionamiento correcto del entorno utilizamos la detección de un objeto común como es un reloj.

En la segunda prueba, que resultará más difícil. Trataremos de detectar una señal de helipuerto. El problema que surge al detectar este objeto es que las señales solo tienen un elemento común, en la mayoría de los casos es la letra “H”, esto hace que el objeto a detectar sea este. Esto complica más la detección, debido a que el objetivo es muy pequeño.

Al ser objeto muy pequeño hay que hacer la ventana de búsqueda muy pequeña, y por tanto el proceso es muy costoso (computacionalmente), especialmente si tenemos imágenes de alta resolución. Normalmente estas técnicas reducen la resolución de la imagen antes de realizar la búsqueda, para reducir el coste, pero si el objeto ya es pequeño, esto dificultaría más la detección.

1. Anexo III. Haar-cascade scripts

Fig. 4. 3 Imágenes positives originales

4.2.2. Creación de imágenes positivas

Como únicamente tenemos una imagen positiva debemos de crear más y para ello emplearemos nuestras imágenes negativas. Pero primero debemos eliminar algunas imágenes dentro de la carpeta de imágenes negativas, estas imágenes las denominaremos como *erróneas*.

¿Qué son estas imágenes? La respuesta es simple, cuando descargamos las imágenes procedente de los enlaces dentro del *synset,* puede suceder que la imagen que descarguemos se haya eliminado o simplemente el enlace no sea válido. Esto provoca que se descargue una imagen por defecto que no nos aporta ninguna información, pero molesta a la hora de realizar el entrenamiento y aprendizaje, ya que estamos incluyendo una imagen más.

Fig. 4. 4 Ejemplo imagen *errónea*

Para eliminar emplearemos un *script* de *python* sobre la carpeta de imágenes negativas. Este *script*[1] necesita de una muestra de imagen *errónea* para saber qué imágenes ha de eliminar.

Una vez tenemos todas las imágenes negativas válidas, empleamos una superposición de nuestra imagen válida sobre las negativas para obtener las imágenes positivas. Para ello primero debemos de crear el descriptor de las imágenes negativas, este descriptor *bg.txt* contiene la lista de imágenes que emplearemos como fondo a la hora de obtener una versión distorsionada de la imagen, es decir, como fondo de la superposición de imágenes entre las negativas y nuestra imagen positiva.

Para obtener este archivo *bg.txt* utilizaremos el *script*[1].

A partir del archivo *bg.txt* somos capaces de obtener las imágenes positivas, para ello ejecutaremos el siguiente comando:

1. Anexo III. Haar-cascade scripts

opencv_createsamples -img positiveimage5050.jpg -bg bg.txt -info info/info.lst -pngoutput info -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5 -num 1800

Con el comando anterior lo que hacemos es mediante *-img* establecemos como imagen fuente nuestra imagen positiva y empleamos como fondo las imágenes descriptivas, esto con indicamos con el descriptor anteriormente creado, *-bg bg.txt*.

Con el atributo *-info* indicamos donde crearemos la lista de imágenes positivas así como la descripción de ellas como podemos ver en la **Fig. 4.8**.

En el archivo *info* podemos ver como en cada línea se indica el nombre de la imagen, el número de objetos detectados dentro que corresponde al primer número después del nombre de la imagen y finalmente se indica la posición del objeto. En la **Fig. 4.8** podemos ver un ejemplo de archivo *info*. En este archivo los últimos cuatro números indican los ejes *X, Y* y el alto y ancho el objeto de la imagen.
A continuación indicamos cuánto se moverá nuestra imagen fuente (imagen positiva) dentro del fondo, esto lo indicamos con la variación en los diferentes ejes, *-maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5.*

```
0001_0014_0051_0027_0027.jpg 1 14 51 27 27
0002_0038_0026_0034_0034.jpg 1 38 26 34 34
0003_0046_0025_0024_0024.jpg 1 46 25 24 24
0004_0016_0009_0067_0067.jpg 1 16 9 67 67
0005_0006_0035_0052_0052.jpg 1 6 35 52 52
0006_0006_0012_0047_0047.jpg 1 6 12 47 47
0007_0030_0034_0057_0057.jpg 1 30 34 57 57
0008_0047_0040_0033_0033.jpg 1 47 40 33 33
0009_0012_0026_0044_0044.jpg 1 12 26 44 44
0010_0028_0040_0055_0055.jpg 1 28 40 55 55
0011_0047_0035_0032_0032.jpg 1 47 35 32 32
0012_0016_0008_0050_0050.jpg 1 16 8 50 50
0013_0018_0009_0066_0066.jpg 1 18 9 66 66
0014_0014_0050_0026_0026.jpg 1 14 50 26 26
0015_0011_0005_0044_0044.jpg 1 11 5 44 44
0016_0019_0048_0038_0038.jpg 1 19 48 38 38
0017_0032_0027_0062_0062.jpg 1 32 27 62 62
0018_0051_0050_0034_0034.jpg 1 51 50 34 34
0019_0026_0039_0036_0036.jpg 1 26 39 36 36
0020_0008_0025_0032_0032.jpg 1 8 25 32 32
0021_0044_0031_0033_0033.jpg 1 44 31 33 33
0022_0046_0015_0026_0026.jpg 1 46 15 26 26
0023_0017_0021_0041_0041.jpg 1 17 21 41 41
0024_0025_0040_0028_0028.jpg 1 25 40 28 28
0025_0015_0006_0037_0037.jpg 1 15 6 37 37
0026_0028_0028_0059_0059.jpg 1 28 28 59 59
0027_0022_0035_0030_0030.jpg 1 22 35 30 30
```

Fig. 4. 5 Ejemplo *info.lst*

Finalmente, mediante el atributo *-num 1800* estamos indicando cuantas muestras positivas queremos crear.

Fig. 4. 6 Ejemplo de imágenes positives creadas a partir de las originales

4.2.3. Entrenamiento del clasificador

Una vez tenemos todas las imágenes necesarias para entrenar nuestro clasificador debemos de crear el archivo *.vec*. Este archivo contiene el nombre de todas las imágenes positivas y la localización del objeto a detectar dentro de la imagen.

Necesitamos crear este archivo porque en el cargaremos la información correspondiente a las imágenes, esto facilita la carga computacional dentro del clasificador. Para crear este archivo emplearemos el siguiente comando:

opencv_createsamples -info info/info.lst -num 1800 -w 20 -h 20 -vec positives.vec

Con este comando estamos empaquetando las imágenes en un vector, archivo .*vec*, por tanto debemos indicar al clasificador donde se encuentran las imágenes positivas (*-info info/info.lst*), el número de imágenes, el tamaño de las imágenes de entrenamiento y las características de las imágenes.

A continuación, debemos de entrenar nuestro clasificador para ello emplearemos el siguiente comando:

opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1900 -numNeg 900 -numStages 15 -w 20 -h 20

Mediante este comando lo que estamos haciendo es indicar, primero de todo donde guardará los resultados, en la carpeta *data* (*-data data*). Después le indicamos el archivo *.vec* que es el que contiene la información de las imágenes positivas, seguidamente añadimos el parámetro para indicar el archivo *bg.txt* donde aparecen las características de las imágenes negativas.

A continuación indicamos el número de imágenes negativas (900 imágenes) y positivas (1900 imágenes) que emplearemos en el entrenamiento, así como su ancho y alto (*-w 20 -h 20*) que debe ser la misma que asignamos en la creación de las muestras.

Por último indicamos el número de etapas de ejecución de nuestro clasificador, hay que tener en cuenta que mientras más etapas ejecuten más preciso es.

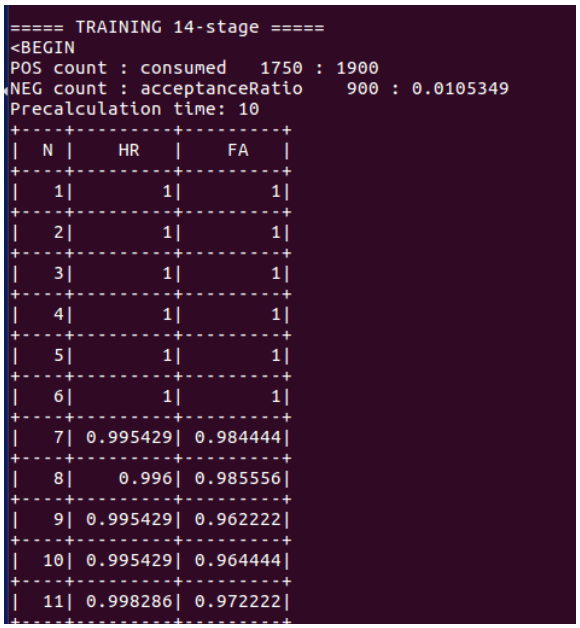

Fig. 4. 7 Entrenamiento del clasificador en 14 *stages*

Aunque mediante este método únicamente podemos emplear una imagen positiva y a partir de ella crear más, existe otra forma de mejorar la capacidad de detección o como mínimo intentarlo.

En este método lo que hacemos es unir mediante un script los archivos *.vec* realizados con dos imágenes positivas diferentes. De esta forma obtenemos un mayor número de imágenes positivas, esto a nivel teórico mejora la capacidad de detección. Aunque se parezca mucho a la forma original de entrenamiento de un clasificador mediante imágenes positivas del objeto a detectar, es totalmente diferente, ya que empleamos un número ínfimo de imágenes positivas y creamos más a partir de ellas.

Esta unión de archivos *.vec* se suele usar para aumentar el número de imágenes positivas en otros métodos. En casos donde tenemos pocas imágenes positivas y queremos aumentar este número con imágenes artificiales.

Las imágenes artificiales son aquellas imágenes que se basan en las imágenes positivas originales con alguna variación, como modificar el contraste, rotar la imagen, etc.

Esta solución no resulta muy acertada porque estamos distorsionando nuestras muestras positivas con otras que no corresponden al entorno real.

Pero este no es nuestro caso. Nosotros estamos empleando en ambos casos imágenes artificiales creadas a partir de una imagen positiva, más adelante comprobaremos si resulta útil o efectiva esta opción.

Ejecutando el comando siguiente creamos un archivo *.vec* a partir de diferentes archivos *.vec* que se encuentran dentro de una carpeta.

```
python mergevec.py -v your_vec_directory -o your_output_filename.vec
```

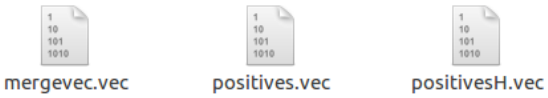

Fig. 4. 8 Directorio donde aplicamos el *merge* de los archivos *.vec* para generar uno que contenga toda la información

4.2.4. Test de precisión del clasificador

Una vez hemos entrenado nuestro clasificador debemos de comprobar su eficacia. Hay que tener en cuenta que el entrenamiento se ha realizado sobre un portátil normal y de gama baja, por lo que el número de etapas no ha sido muy elevado. Una opción es emplear una máquina virtual y preparar el entorno dentro de ella, esto conlleva un cierto gasto.

Para testear el clasificador hemos empleado dos formas. Una mediante la cámara del portátil la cual captura imágenes a tiempo real y podemos ver en la pantalla si el objeto ha sido detectado, y otra vía es aplicar el clasificador sobre una carpeta con imágenes de testeo.

Para realizar estas pruebas hemos empleado diversos scripts[1] uno en el cual empleamos un conjunto de imágenes de test**,** y otro en el cual realizamos una detección a tiempo real empleando la webcam del portátil.

En el segundo caso ha sido imposible obtener una imagen de muestra en la cual se vea el objeto detectado. El principal problema es la dificultad a la hora de detectar el objeto, pero esto puede ser debido a que empleábamos la cámara del portátil y la calidad de la imagen no era lo suficientemente buena.
Cuando hemos utilizado imágenes de testeo hemos obtenido diferentes resultados. Ninguno de ellos ofrece unas buenas expectativas.

Fig. 4. 9 Resultados con prueba de detección de relojes

En la primera prueba, el caso referente al reloj **Fig. 4.14** hay cierto error pero funciona correctamente, como se ha comentado anteriormente hay que tener en cuenta las herramientas empleadas para el entrenamiento del clasificador.

1. Anexo III. Haar-cascade scripts

Por otro lado, en el segundo caso **Fig. 4.15** y más complicado la detección tiene muchos errores y problemas, aunque consigue detectar en algunos casos el objeto. El número de falsos positivos es demasiado elevado. Esto era algo que preveíamos que pudiera pasar. Debido a que el objeto a detectar es más pequeño y la venta ventana de búsqueda también. Además de sumarle la capacidad computacional del portátil.

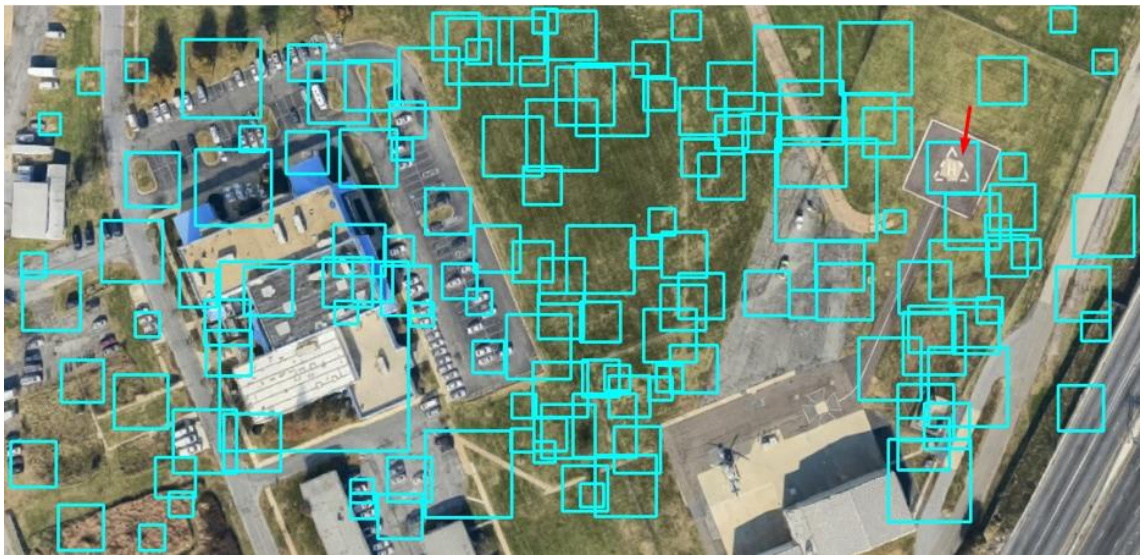

Fig. 4. 10 Resultado prueba de detección de señal de helipuerto (señal marcada con flecha roja)

En la **Fig. 4.16** podemos ver un ejemplo empleando el archivo .*vec* creado por el script merge. La principal diferencia es el número de falsos positivos, pero persiste el gran número de errores.

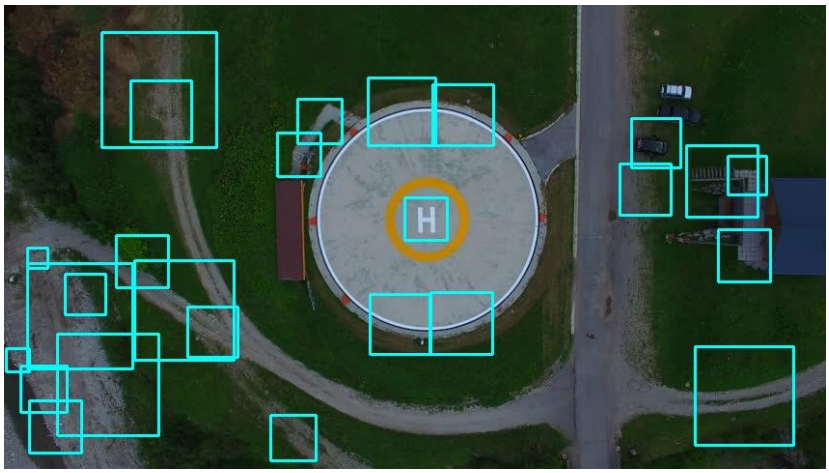

Fig. 4. 11 Resultado de la prueba de detección con fichero .vec realizado con merge

Ahora analizaremos con más detalle los resultados obtenidos.

Tabla 4. 1 Resultados testeo haar-cascade

Objeto	Método	Nº de imágenes (muestras)	Falsos positivos	Positivos reales
Reloj	Sin merge	100	124	38
	Con merge	100	0	3
Señal de helipuerto 1	Sin merge	40	2534	5
	Con merge	40	2417	5
Señal de helipuerto 2	Sin merge	15	178	1
	Con merge	15	235	3

En la **Tabla 4.1** podemos apreciar que hay dos casos de la señal de helipuerto. Esto es debido a que hemos empleado dos tipos de imágenes ligeramente diferentes. En el caso dos la imagen es con un enfoque más cercano a la señal de helipuerto.

Fig. 4. 12 Imágenes señales helipuerto (caso 1 a la izquierda y caso 2 a la derecha)

Analizando los resultados de la **Tabla 4.1** podemos sacar diferentes conclusiones[1].

En el caso de los relojes podemos distinguir dos resultados. El primer resultado sin aplicar merge. En este caso haar-cascade es capaz de detectar 38 positivos reales, 62 falsos negativos y 124 falsos positivos del total de 100 muestra.

Esto nos da una precisión del 23.46%. Pero si analizamos la sensibilidad (*recall*) obtenemos que es del 38%.

En el segundo resultado utilizando merge no hay falsos positivos. Esto nos da una precisión del 100%. Pero por otro lado la sensibilidad es del 3%.

Si analizamos los resultados de la señal de helipuerto nos encontramos con resultados más dispersos, ya que hemos realizado dos casos.

En el primer caso y sin utilizar merge obtenemos una precisión del 0.2% debido al gran número de falsos positivos. Si estudiamos la sensibilidad esta es del 12.5%.

En este primer caso utilizando merge la precisión es del 0.2% también debido al gran número de falsos positivos. Respecto a la sensibilidad, esta no varía en

1. Anexo III. Evaluación del rendimiento de un clasificador

comparación con no usar merge. Debido a que el número de detecciones es la misma en ambos casos.

En el segundo caso empleando imágenes con un enfoque más cercano obtenemos otros resultados. Primero sin merge la precisión sube hasta 0.5% y la sensibilidad baja hasta el 6.6%.

Si analizamos este caso empleando merge nos encontramos con una precisión del 1.26%. Y respecto a la sensibilidad esta sube hasta el 20%, bastante más elevada que en los otros casos.

4.2. Conclusiones

Después de haber realizado diversas pruebas a diferente número de *stages,* ya que la capacidad computacional del portátil no es muy elevado y surgían algunos problemas, hemos obtenidos dos tipos de resultados.

Unos dependen del objeto a detectar, como hemos podido ver la capacidad de detección del reloj ha sido mucho más eficaz y significativa que a la hora de detectar la señal de aterrizaje de un helicóptero.

Por otro lado, otro punto importante son los resultados obtenidos empleando el script *merge*. En el caso de los relojes hemos obtenido una precisión del 100%, pero una sensibilidad del 3%, es decir, solo ha detectado el reloj en tres imágenes de las 100 que hemos usado para testear el clasificador.

En el caso de las señales de helipuerto utilizar merge reduce el número de falsos positivos. Pero de todas formas sigue siendo muy elevado.

Por todo esto podemos concluir que a la hora de utilizar la unión de los ficheros *.vec* dependerá de la dificultad que suponga detectar este objeto, es decir, con una vista preliminar podemos ver que identificar un reloj es mucho más sencillo que una señal de aterrizaje de helicóptero. Este es un factor fundamental que nos permite discernir entre emplear un método u otro.

También analizando los resultados numéricos nos encontramos con el último caso. En este caso usamos 16 imágenes de test y con un enfoque más cercano, además de usar merge. En él obtenemos una sensibilidad del 20%. Pero hay que tener en cuenta que en las imágenes donde detectamos de forma correcta la señal hay un gran número de falsos positivos, por tanto tenemos una precisión muy baja (1.26%).

Este es otro factor a tener en cuenta, el número de falsos positivos.

CAPÍTULO 5. DETECCIÓN DE OBJETOS CON TRANSFER LEARNING

5.1. Deep learning

Deep learning es un campo dentro de *machine learning* que se vale de algoritmos inspirados en la estructura y funcionamiento del cerebro humano, creando redes neuronales artificiales. En si se trata de una enorme red neuronal artificial definidas por diferentes capas y donde emplea una gran cantidad de datos para entrenar.

Un punto importante es que utiliza múltiples transformadas no lineales. Este tipo de transformadas consisten en al alterar la posición de los pixeles pero mantienen la intensidad de ellos. Esto nos permite distinguir un objeto en diferentes posiciones y establecer una etiqueta.

Además deep learning cuenta con bases de datos inmensas donde acumula esta información.

Esto conforma un sistema de gran capacidad que nos permite por ejemplo, que si analizamos la imagen de un coche, no será necesario convertir la imagen en un vector y partir de que simplemente se trata de una imagen.

Esto quiere decir que habrá una etiqueta que identifique inmediatamente esta imagen, y sabrá que se trata de un coche. Debido que al contener tanta información y haber entrenado tanto ya sabrá ciertas cosas.

Este nuevo sistema de aprendizaje a diferencia de los algoritmos tradicionales sigue aprendiendo a medida que más datos posean, siendo escalable y versátil.

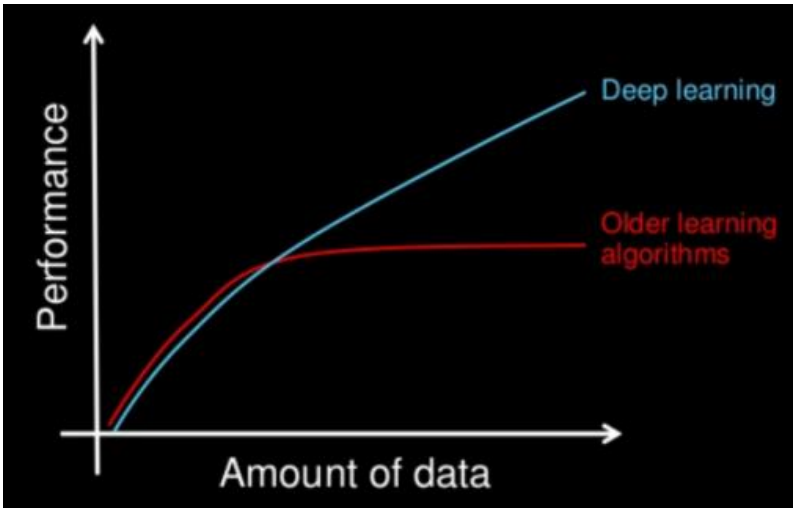

Fig. 5. 1 Capacidad de aprendizaje a partir de la cantidad de información [50]

Un factor importante en el tipo de datos que empleamos en *deep learning* es que se trata de datos etiquetados o por aprendizaje supervisado. Pero esto no es algo negativo, ya que la gran cantidad de datos que posee y poseerá la red permitirá analizar información mediante un método no supervisado sin problemas, simplemente es cuestión de tiempo.

Otro punto importante es que se pueden incorporar nuevos algoritmos según los datos a tratar y nuevos modelos a partir de la información adquirida, como hemos comentado se trata de un crecimiento constante.

Esta forma de aprendizaje por capas nos permite obtener características a diferentes niveles y por tanto mejorar la detección de objetos. Este análisis más detallado de las características obliga a emplear elementos con una capacidad mayor de procesado.

A consecuencia de esto se utilizan procesadores con una mayor capacidad o se busca combinar diferentes formas de realizar las tareas. En resumen se emplean GPU (*Graphicsl Processor Unit*) para el análisis de los datos que ofrecen núcleos que realizan operaciones iterativas en paralelo mejorando el rendimiento, más adelante en la fase experimental explicaremos con más detalle las ventajas de estos procesadores ofrecen.

Fig. 5. 2 Representación de características de una imagen a diferentes niveles

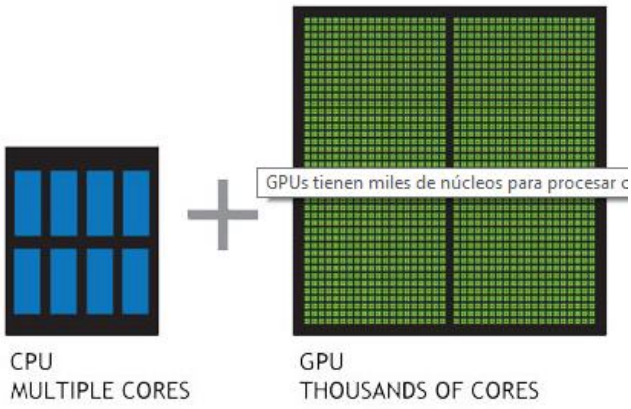

Fig. 5. 3 Comparativa de núcleos en CPU y GPU [51]

5.2. ¿Qué es CNTK (Cognitive Toolkit)?

Se trata de un *framework* de *deep learning* desarrollado por el departamento de investigación de Microsoft, el cual describe diversos tipos de redes neuronales.

Dentro de este *framework* encontramos una gran cantidad de recursos para diferentes lenguajes. También consta con una arquitectura propia.

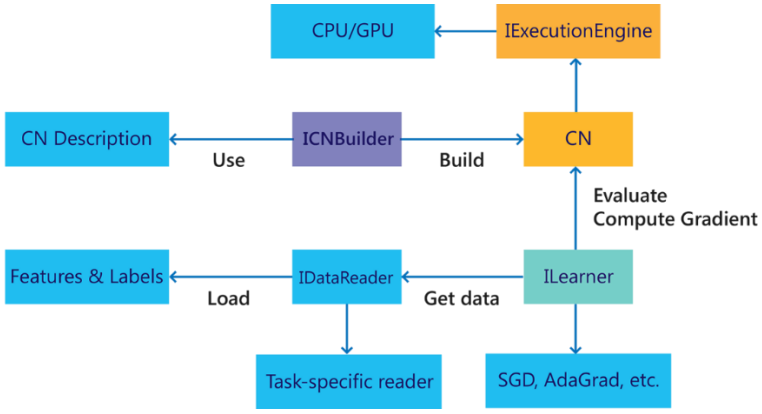

Fig. 5. 4 Diagrama de la arquitectura de CNTK [49]

Por otro lado, cómo trabajaremos empleando *python* como lenguaje de desarrollo utilizaremos la librería de *python* que ofrece CNTK donde incluye modelos de definición y computación, algoritmos de aprendizaje y lectura de datos y aprendizaje distribuido.

CNTK ofrece principalmente diferentes modelos neuronales y otro tipo de modelos también de gran utilidad:

- *Feedforward Deep Neural Networks (multilayer perceptron – MLPs[1]):* Este tipo de modelos se diferencia de una red neuronal típica en el número de capas que contiene, por ello se le profunda.

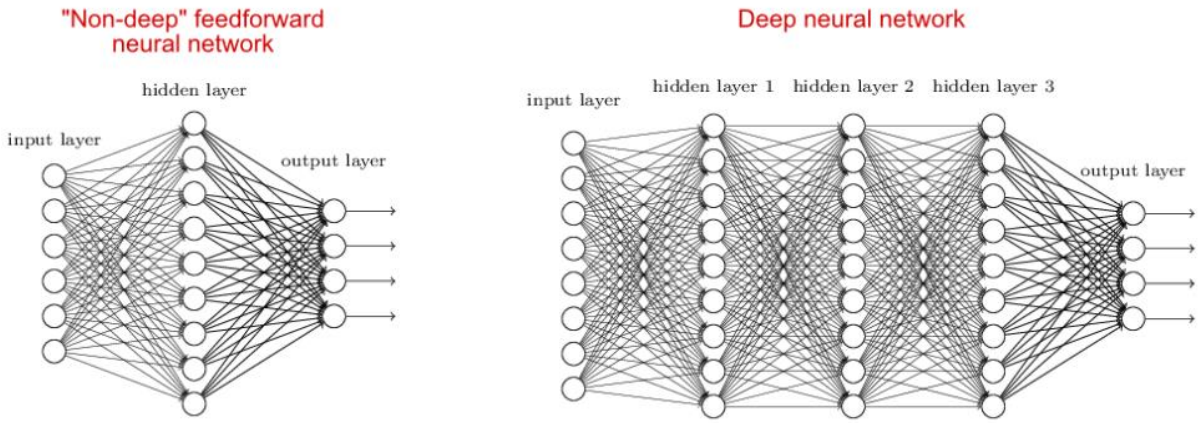

Fig. 5. 5 Comparación entre una red neuronal y una red neuronal profunda [49]

Otro factor es el concepto *feedforward* que indica que esta redes la información fluye en un sentido no hay procesos cíclicos entre los diferentes núcleos y capas, de manera que no hay conexiones de *feedback.*

1. Anexo I: Multilayer perceptron

- *Recurrent Neural Networks (RNN) / Long Short Term Memory Units (LTSM):* Estos dos modelos están juntos porque *LSTM* es un tipo de *RNN* el cual incluye una célula de memoria que permite almacenar información por periodos largos. La principal característica de las *RNN* es que tienen procesos cíclicos entre núcleos. Esto da la capacidad de emplear su memoria interna para procesar secuencias arbitrarias de entrada, de manera que obtienen tanto un *feedback* como un *feedforward*.

- *Stochastic Gradient Descent (SGD):* Este modelo sigue el proceso de cualquier clasificador que emplee el *Gradient Descent,* pero a diferencia de éste en cada muestra se actualiza el error para mejorar la siguiente predicción, de manera que evalúa y actualiza los coeficientes en cada iteración.

- *Convolutional Neural Networks (CNN):* Este tipo de modelo funciona exactamente igual que una red neuronal típica la principal diferencia es que asumen que la entrada que recibirán se trata de una imagen, por ello emplearemos este clasificador en el proyecto. En el siguiente apartado explicaremos con más detenimiento su funcionamiento y características.

5.3. Convolutional Neural Networks (CNN ó ConvNets)

Antes de explicar el funcionamiento y que son las redes CNN debemos de decir que no utilizamos una red CNN, sino que empleamos una *Deep Convolutional Neural* Network (DCNN). Su funcionamiento y características son casi iguales, la única diferencia reside en que ofrece una mayor profundidad, es decir, mayor número de capas y mejor análisis.

Las redes neuronales convolucionales son un tipo de red neuronal artificial que se diferencia por su gran efectividad a la hora de reconocer y clasificar imágenes, ya que fue creado con la idea de que recibiera como entrada imágenes. Este tipo de redes neuronales actualmente son muy eficientes a la hora de detectar caras, objetos y señales de tráfico, por ello forman parte en la robótica para potenciar la visión de los robots y para el piloto automático de los coches.

A continuación veremos cómo funcionan las redes neuronales convolucionales y los diferentes procesos que llevan a cabo para realizar una clasificación y aprender.

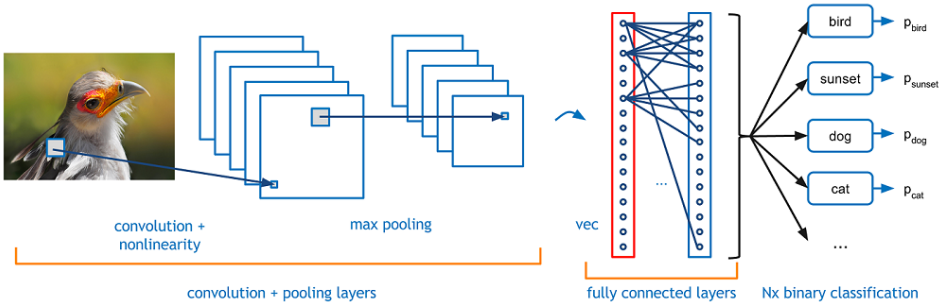

Fig. 5. 6 Diagrama de un clasificador CNN [66]

A partir de la **Fig. 5.6** podemos ver la estructura del funcionamiento de una *ConvNets* divida en dos bloques principales y una parte final donde se encuentran las diferentes etiquetas o elementos clasificables.

Una vez realizada la clasificación, la suma de las probabilidades de todas estas etiquetas es igual a uno. Partiendo de esto vamos a ver los diferentes procesos dentro de estas redes. Estos procesos no se efectúan directamente en el orden que son explicados, ya que se trata de bloques que pueden emplear de forma iterativa.

Antes de empezar con los diferentes bloques que componen las redes neuronales convolucionales hemos de tener en cuenta que las imágenes pueden ser representadas como una matriz donde cada píxel tiene un valor determinado.

En el caso de las imágenes a color se utilizan tres canales (rojo, azul y verde). Por otra parte en las imágenes en escala de grises simplemente empleamos un canal de manera que resulta más fácil de tratar, por ello antes trabajar con una imagen se suele realizar esta conversión a escala de grises.

Primero veremos el primer gran bloque el cual se denomina la capa de convolución y agrupación, donde se encuentran los siguientes bloques:

5.3.1. Convolución

Las redes neuronales convolucionales tienen este nombre en parte debido a este proceso (convolución). En la convolución partimos de la matriz de la imagen que estará compuesta por valores que van de 0 a 255, sobre esta imagen aplicaremos un filtro que recorre esta imagen y nos da como resultado una característica de esta imagen.

1	0	1	1	1
0	0	1	1	0
0	1	1	0	1
1	1	1	0	0
1	0	0	0	1

0	1	0
1	0	1
0	1	0

Fig. 5. 7 Representación de la matriz de la imagen (izquierda) y de la matriz del filtro (derecha)

Aplicando el filtro sobre la imagen realizamos la multiplicación del valor de los píxel con el filtro. Este filtro también es denominado *kernel* ó *feature detector*

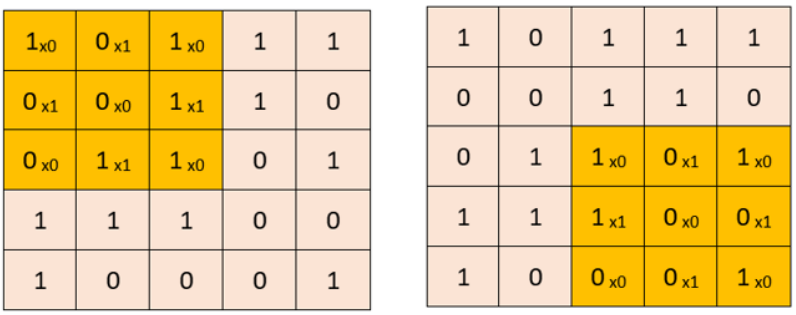

Fig. 4. 13 Representación de aplicar el filtro sobre la matriz de la imagen (primera y última posición del filtro)

Como resultado de la aplicación del filtro obtenemos una matriz que corresponderá al resultado de la suma de las multiplicaciones, y que tendrá una dimensión que dependerá del recorrido que realice el filtro. Esta matriz se denomina *feature map* y es una característica de la imagen.

2	2	2
2	3	3
3	2	1

Fig. 5. 8 Matriz resultante de aplicar el filtro

Este sería el proceso de convolución pero en él intervienen diferentes elementos que hacen que sea más complejo, sin tener en cuenta todos los procesos matemáticos que se realizan en segundo plano.

Por un lado hay diversos tipos de filtros que podemos utilizar. El utilizar diferentes filtros nos permite identificar diferentes características y rasgos de la imagen, como curvas, bordes, etc.

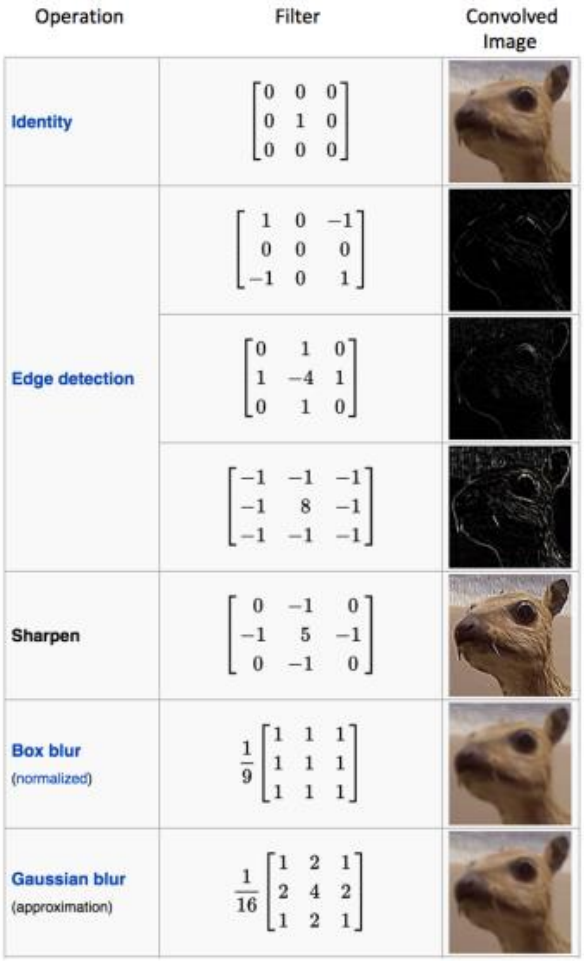

Fig. 5. 9 Ejemplos de filtros [64]

Siguiendo con el concepto de filtros surge otro que es el de profundidad (*depth*) que indica el número de filtros empleados en el proceso de convolución. La profundidad también indica el número de *feature maps* que obtendremos de la imagen, ya que por cada filtro que aplicamos obtendremos una.

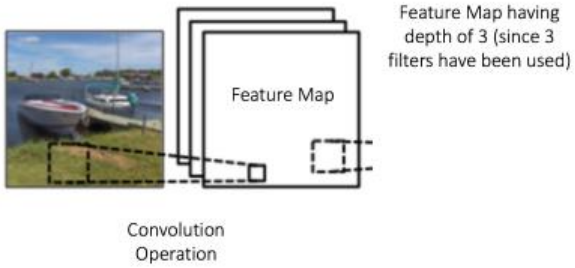

Fig. 5. 10 Ejemplo de profundidad [64]

Otro parámetro importante en el bloque de convolución es el *stride*. Este parámetro indica cada cuántos píxeles se aplica el filtro.

En el ejemplo que podemos ver en las **Fig. 5.7**, **Fig. 5.8** y **Fig. 5.9** utilizábamos un *stride* de uno, de manera que cada un píxel aplicamos el filtro hasta que

cubrimos toda la imagen. En algunos casos se aplicaba el filtro varias veces sobre el mismo píxel.

Normalmente se aplica un *stride* de uno, pero utilizar uno mayor nos permite construir un modelo que se comporta como una red neuronal recursiva.

Como último elemento importante encontramos el *padding*.

A veces es interesante aplicar un *padding* por el borde de la matriz de entrada. Esto nos permite tener un control del tamaño de las *features maps*, es decir, obtenemos un elemento de la misma longitud a la entrada y a la salida. Esto a nivel matricial supone una mayor agilidad en los cálculos.

Aplicar padding se denomina *wide-convolutional* y el no aplicarlo se conoce como *narrow convolutional*.

5.3.2. Funciones de activación (*Activation functions*)

Este bloque de operaciones se realiza después de cada bloque de convolución. Hay diferentes tipos de funciones de activación, pero todas tienen el mismo objetivo.

El objetivo de estas funciones es hacer que la función no sea lineal. Si en cada nodo obtenemos una salida lineal y en cada etapa obtenemos una salida lineal estaremos representando el comportamiento de un nodo linear, de manera que no tiene sentido tener más nodos, con un único nodo bastaría.

También se emplea para reducir la variación entre los extremos de la función, de manera que obtenemos una función no lineal y más aplastada, con los extremos más cercanos. Existen diferentes tipos de funciones de activación:

- **Tanh:** Esta función hace que los valores recibidos por un set de entradas quede en el rango de [-1, 1]. La función que representa es *f(x) = 1/(1 + exp(-x)).*

- **Sigmoid:** Esta función que se encuentra dentro de la función *tanh* reduce el rango de valores para que sea de [0, 1]. Las imágenes se obtienen con la función *f(x) = (2/(1 + exp(-x))) - 1.*

- **ReLU (Rectified Linear Unit):** Esta función simplemente aplica un límite inicial, de manera que todos los valores negativos pasan a ser 0, de manera que la función es *f(x) = max(0,x)*.

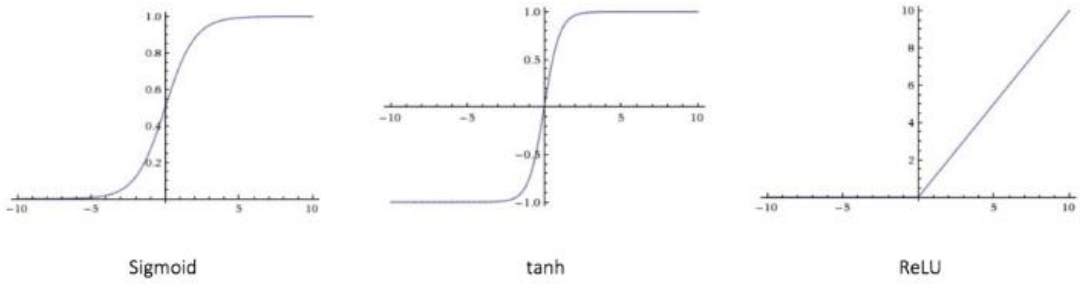

Fig. 5. 11 Representación de las funciones de activación [57]

En la siguiente imagen podemos ver una ejemplo visual de cómo cambia una imagen al aplicar una función de activación, en este ejemplo aplicamos la función *ReLU.*

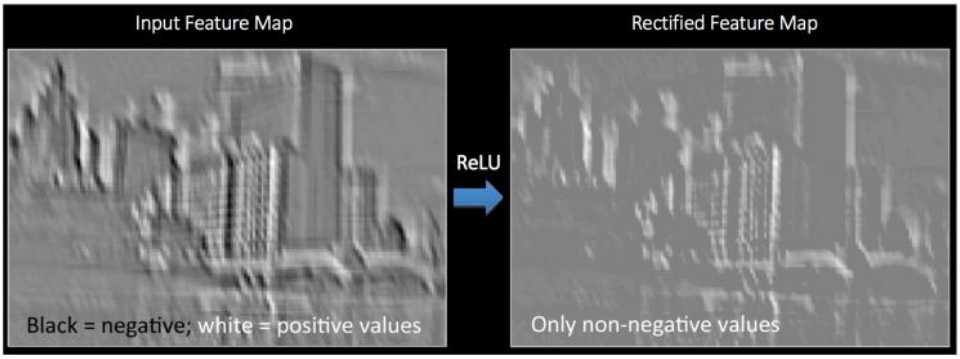

Fig. 5. 12 Ejemplo de aplicación de la función ReLU [60]

5.3.3. Agrupación (*Pooling*)

La agrupación espacial o también denominada *subsampling* o *downsampling* sirve para reducir la dimensión de cada *feature map* y quedarse sólo con lo más importante. Esta función se aplica siempre después de haber efectuado una función de activación como puede ser *ReLU*.
Para realizar la agrupación se crea una ventana para agrupar las muestras, a partir de esta ventana se aplican diversas formas de seleccionar de la información:

- **Max Pooling:** Mediante esta forma se reduce el número de *features maps* sólo escogiendo la característica de mayor valor dentro de la ventana.

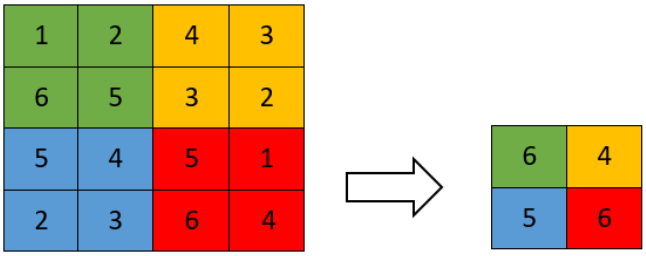

Fig. 5. 13 Ejemplo *Max Pooling*

- **Sum Pooling:** Este método aplica la suma como operación. El resultado de la suma se compone por todos los valores de las características que hay en la ventana.

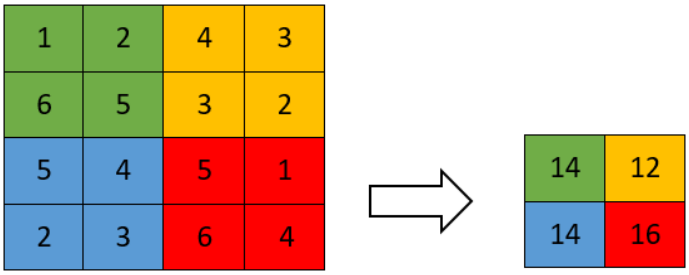

Fig. 5. 14 Ejemplo *Sum Pooling*

- **Average Pooling:** Utilizando esta manera de agrupar las características lo que obtenemos es la media aritmética de los valores que se encuentran dentro de la ventana.

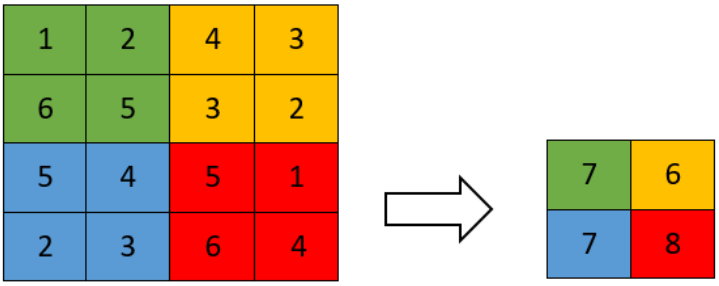

Fig. 5. 15 Ejemplo *Average Pooling*

En la agrupación espacial la ventana se desplaza dependiendo del tamaño de la ventana, es decir, si la ventana es de 2x2 se desplaza cara dos celdas, de manera que si aplicamos una ventana de *NxN* el desplazamiento de la ventana es de *N*.

Las características principales que ofrece y por qué se aplica la agrupación espacial es:

- Hace que las características de entrada, correspondientes a las imágenes, más pequeñas, menor cantidad, y más manejables.

- Al reducir el número de características se consigue una reducción de parámetros y cálculos dentro de la red, de manera que se obtiene un control del sobreajuste.

- Aporta un cierto control de errores, haciendo la red invariante a pequeñas transformaciones, distorsiones o traslaciones en la imagen de entrada. Esto es debido a que empleamos métodos como el *max pooling, sum pooling* y *average pooling* en una ventana local.

- Conseguimos una representación a escala casi invariante de nuestra imagen de entrada, esto nos permite localizar objetos dentro de la imagen independientemente de donde se encuentren dentro de la imagen.

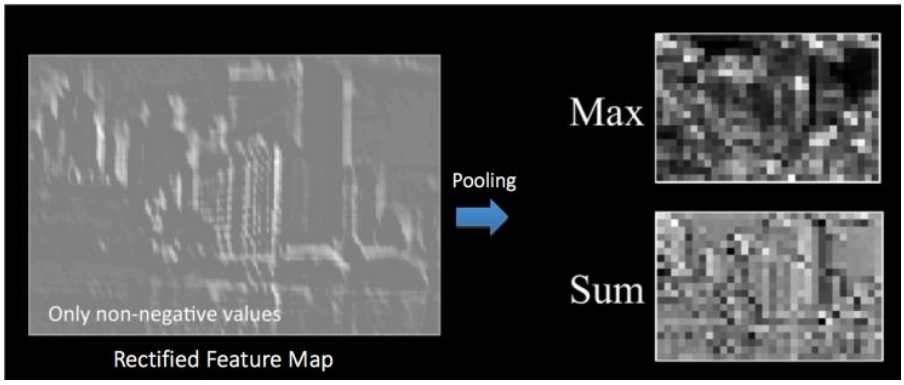

Fig. 5. 16 Ejemplo y comparación entre *max pooling* y *sum pooling* [60]

5.3.4. Capa totalmente conectada (*fully connected layer*)

Después de haber visto el gran bloque anterior que corresponde a la capa de convolución y agrupación, ahora analizaremos el siguiente gran bloque que es donde se interconectan los diferentes núcleos (*neuronas*) que componen nuestra red convolucional neuronal.

Una vez hemos obtenido las diferentes *features maps* correspondientes a la imagen de entrada, debemos de realizar una clasificación de lo que estamos analizando.

Para realizar la clasificación aplicamos un clasificador denominado *MLP* (*Multilayer Perceptron*)[1].

Este clasificador interconecta mediante nodos (*neuronas*) diferentes características (*features maps*) obtenidas anteriormente y aplica unos pesos sobre ella para poder así clasificar el objeto.

De esta forma la última capa contiene los nodos que representan las etiquetas.

Esta clasificación no se aplica una única vez. Como hemos comentado antes se trata de una capa donde después de aplicar el clasificador se vuelve a aplicar una función de activación, como por ejemplo *ReLU*, esto es debido a que el clasificador trabaja con entradas no lineares.

Todo este proceso se realiza porque aunque se pueda obtener una buena clasificación a partir de las características obtenidas anteriormente, el combinarlas mejora esta clasificación.

1. Anexo I: Multilayer perceptron

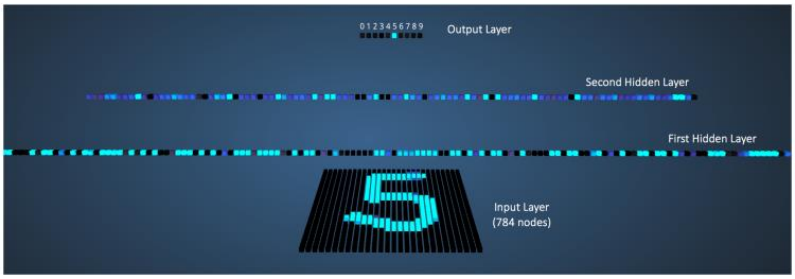

Fig. 5. 17 Ejemplo visual del funcionamiento y clasificación son MLP [57]

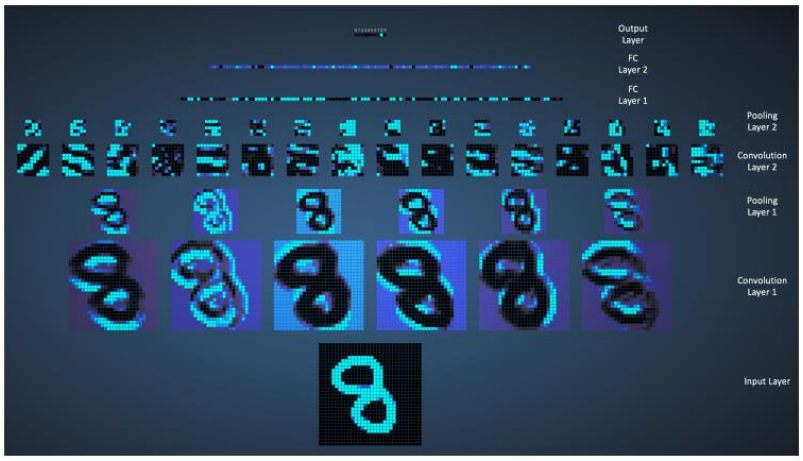

Fig. 5. 18 Ejemplo visual de las diferentes capes y funcionamiento de CNN [64]

5.4. Transfer learning

Hoy en día, aun contando con grandes bases de datos y con datasets de gran tamaño sigue habiendo el problema. Este problema es que hay objetos o elementos de los cuales no podemos realizar una detección correcta debido a la falta de imágenes de muestra. Pero actualmente, las redes neuronales profundas mediante el aprendizaje consiguen un mapeo detallado tanto de la entrada como de la salida.

El funcionamiento de esta transferencia de conocimientos no obtiene la precisión de clasificación que se obtiene con el método tradicional supervisado, pero la principal ventaja es que nos permite ahorrarnos el gran coste que supone obtener un dataset con imágenes suficiente para una clasificación correcta.

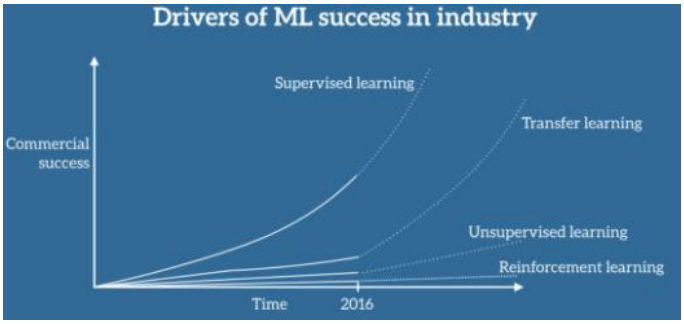

Fig. 5. 19 Comparación entre métodos de clasificación a nivel comercial [72]

Mediante el método tradicional entrenamos nuestro clasificador con imágenes pre-etiquetadas y de esta forma creamos un modelo, el cual testeamos con otro conjunto de imágenes. Todas las imágenes tanto de entrenamiento como testeo son del mismo dominio.

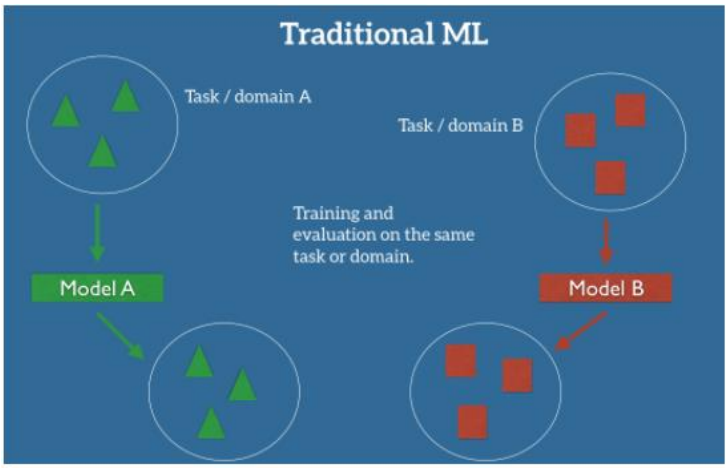

Fig. 5. 20 Diagrama de aprendizaje mediante método tradicional [72]

Utilizando la transferencia de conocimiento lo que realizamos es un entrenamiento con imágenes de un dominio que no es el objetivo, obteniendo un mapeo o modelo.

Este conocimiento adquirido en la generación del modelo se aplica para crear otro modelo que pertenece a otro conjunto de imágenes de otro dominio.

Finalmente el nuevo modelo se testea sobre un conjunto de imágenes del nuevo dominio, que el objetivo.

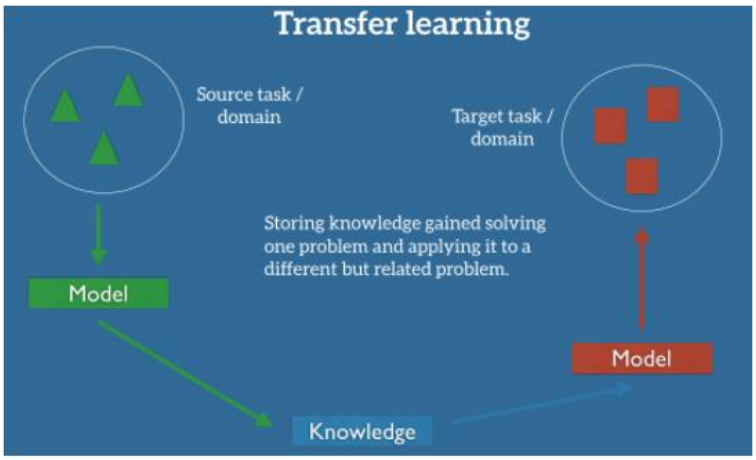

Fig. 5. 21 Diagrama de aprendizaje mediante transfer learning [72]

5.4.1. Transferencia de conocimientos en DCNN

En una transferencia de conocimientos, como hemos comentado anteriormente partimos de un modelo que es entrenado por una gran cantidad de imágenes

etiquetadas. A partir del entrenamiento previo obtenemos las diferentes capas que conforman nuestra red neuronal convolucional incluyendo la capa totalmente conectada.

Excluyendo la última capa (*fully connected layer*) tenemos el resto de capas que nos permiten obtener las características de las imágenes, las cuales empleamos para distinguir los diferentes objetos dentro de una imagen.

Utilizando las capas obtenidas en el entrenamiento anterior, entrenamos el clasificador con las muestras que poseemos de nuestro dominio objetivo. Estas muestras son insuficientes para una detección eficiente utilizando el método tradicional.

Pero con este nuevo entrenamiento conseguimos aprender nuevas características concentradas, con concentradas nos referimos a que cuando aplicamos las capas del clasificador pre-entrenado reducimos el número de características que puede ser demasiado elevado.

Un ejemplo de característica concentrada es el índice de masa corporal donde se establece una relación entre la altura y el peso, aplicando este tipo de relaciones obtenemos una reducción de las características y conseguimos que nuestro clasificador es más eficiente.

Básicamente lo que obtenemos con el primer entrenamiento es un generador de características.

Finalmente debemos de añadir las últimas capas (*fully connected layer*) del clasificador.

Estas capas son las que indican el objeto u objetivo que queremos detectar, es donde se establecen las relaciones entre las características, seleccionando cuales son útiles para la detección.

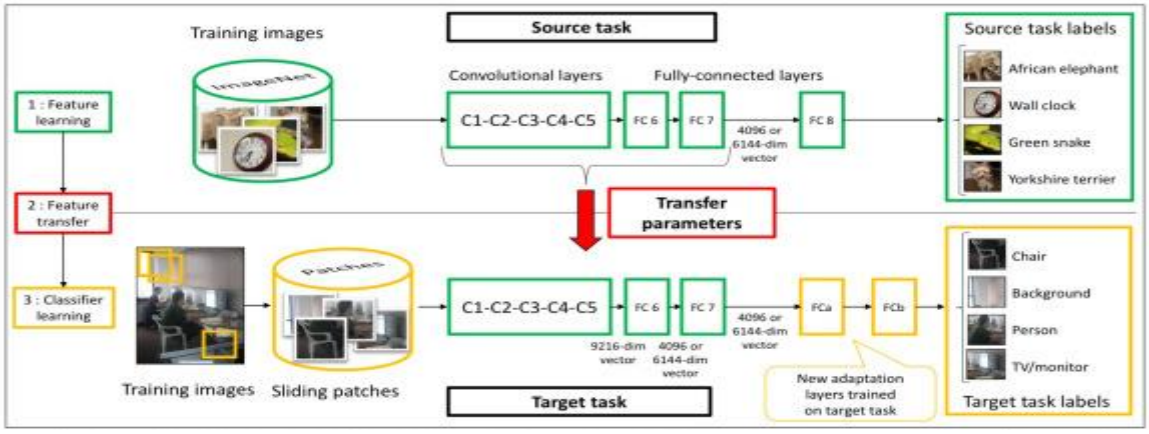

Fig. 5. 22 Diagrama de pasos y elementos en transfer learning [70]

CAPÍTULO 6. FASE EXPERIMENTAL II: TRANSFER LEARNING

En esta segunda fase se han realizado diversas pruebas. Por un lado hemos realizado el ejemplo que propone Microsoft en su página web [73] para comprobar el correcto funcionamiento

Por otro lado hemos realizado las pruebas con imágenes propias utilizadas anteriormente, las señales de aterrizaje de helicópteros y relojes.

6.1. Entorno experimental

Para esta segunda fase experimental la preparación del entorno de trabajo ha sido totalmente diferente a la primera, en este caso hemos necesitado de herramientas externas y virtuales.

6.1.1. Azure

Para la preparación del entorno experimental hemos empleado los servicios de virtualización de Azure. Pero antes de comentar qué herramientas hemos empleado definamos brevemente qué es Azure.

Fig. 6. 1 Icono corporativo de Azure [75]

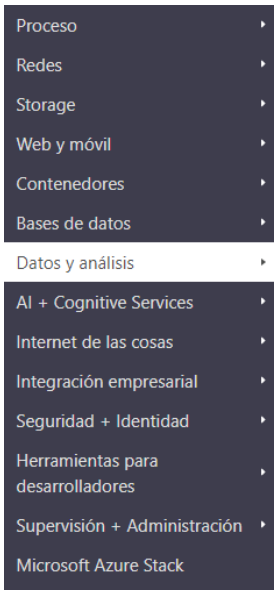

Fig. 6. 2 Servicios Azure [75]

Azure es una plataforma que ofrece servicios en la nube para desarrolladores y profesionales de la industria IT. Estos servicios permiten construir, desplegar y gestionar aplicaciones a través de una red global de *datacenters*, integrando diferentes herramientas y soportes.

No se trata de una simple plataforma de soporte en la nube, ya que ofrece un conjunto de herramientas que van desde *applcations insights* (herramientas para el diagnóstico de páginas webs) hasta servicios de *bots* y *machine learning*.

Se trata también de un servicio multiplataforma que nos permite trabajar desde cualquier lugar, de la misma manera también es multiusuario permitiéndonos trabajar con cualquier sistema operativo y obviamente podemos trabajar con cualquier lenguaje.

6.1.2. Deep Learning Virtual Machine (DLVM)

Para realizar las diversas pruebas de detección de objetos empleando la técnica de *transfer learning* hemos necesitado de un recurso concreto que ofrece Azure.

Este recurso de denomina *Deep Learning Virtual Machine*, que la principal ventaja y elección de esta herramienta es el poder emplear unidades de procesamiento gráfico (GPU).

Las GPU son unidades de procesamiento dedicados y permiten trabajar con una gran cantidad de datos, así como realizar operaciones una y otra vez. El hecho de que son dedicadas es la principal diferencia con la unidad de procesamiento central (CPU), la cual como indica su nombre es una unidad con un propósito general.

Al ser una máquina virtual basada en GPU está preparada para entrenar modelos de *deep learning.* Y por tanto consta con los frameworks más populares y utilizados para este tipo de funciones.

Frameworks como Microsoft Cognitive Toolkit, TensorFlow, Keras, Caffe2, Chainer; herramientas para la obtención y pre-procesado de imágenes, textual data, herramienta para el modelado y desarrollo de datos para actividades como Microsoft R Server Developer Edition, Anaconda Python, Jupyter notebooks for Python and R, IDEs for Python and R , SQL database y otras herramientas para tratar datos así como ML tools.

6.1.3. Despliegue de DLVM

A la hora de desplegar el entorno de trabajo hemos necesitado de una suscripción de Azure. Como el gasto que realizaríamos no sería muy elevado hemos optado por una de suscripción de pago por uso reduciendo así los costes.

Una vez obtenida la suscripción accedemos a los recursos desde el *portal* de Azure para así poder utilizar las diferentes herramientas que nos proporcionan.

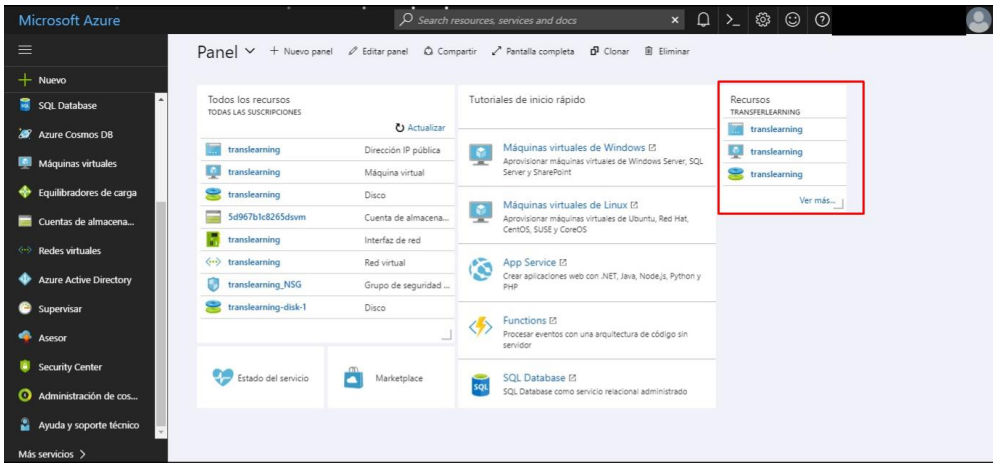

Fig. 6. 3 Pantalla principal del *Portal* de Azure [78]

En la **Fig. 6.3** podemos ver enmarcado en rojo los recursos relacionados con DLVM, ya que Azure como oferta nos ofrece otros servicios gratuitos.

DVLM no es simplemente una máquina virtual sino que incluye otros elementos que aportan un soporte perfecto para realizar tareas de *deep learning*. Hay que tener en cuenta que para las pruebas que necesitamos realizar hemos optado por el servicio más barato y sencillo, ya que no necesitamos unas altas prestaciones. Aun así el servicio ofrece 6vcpu con unos 56GB de memoria. En la **Fig. 6.4** podemos ver los elementos que conforman este servicio.

NOMBRE	TIPO
5d967b1c8265dsvm	Cuenta de almacenamiento
translearning	Disco
translearning	Máquina virtual
translearning	Interfaz de red
translearning	Dirección IP pública
translearning	Red virtual
translearning_NSG	Grupo de seguridad de red
translearning-disk-1	Disco

Fig. 6. 4 Elementos de DLVM [78]

6.2. Desarrollo experimental

6.2.1. Obtención de las imágenes

A la hora de seleccionar las imágenes hemos de establecer dos grupos, unas serán imágenes para entrenar nuestro clasificador y otras serán imágenes para testear el funcionamiento de este.

La principal diferencia entre estos dos grupos es el número de imágenes que tendremos de cada. También hay que añadir que hay un grupo de imágenes dentro de la carpeta test que no son de los objetos que queremos detectar sino que son pruebas de error para ver el comportamiento del clasificador.

Hay que resaltar que para este método no son necesarias imágenes negativas, ya que nuestro clasificador intentara clasificar y diferenciar cualquier imagen de entrada entre las opciones que le demos. En la **Fig. 6.5** podemos ver que las clases que tenemos como baremo son señal de helipuerto y relojes.

Por los resultados podremos ver si se trata de una imagen de un reloj, de una señal de helipuerto o de otra queno corresponde con ninguna de las dos.

Fig. 6. 5 Contenido de la carpeta Test

Primero de todo debemos obtener imágenes de los objetos que queremos detectar, que son aquellos de los cuales poseemos muy pocas imágenes. Para ello emplearemos ImageNet al igual que en el fase experimental con Haar-cascade.

Aunque para entrenar nuestro clasificador empleemos más imágenes que para testear. No utilizamos un número muy grande de imágenes, sino más bien reducido, ya que es al problema al que nos enfrentamos en este proyecto.

En la **Fig. 6.6** podemos ver como utilizamos 5 imágenes de test y 15 de entrenamiento. Un total de 20 imágenes, un número muy reducido.

Fig. 6. 6 Comparativa entre número de imágenes de entrenamiento y de test

Ahora la pregunta que nos planteamos es si con este número tan reducido de imágenes es posible obtener algún tipo de resultado óptimo. Bueno aun siendo el número de imágenes del elemento objetivo muy reducido, el clasificador como hemos explicado anteriormente es entrenado sobre otro grupo de imágenes y únicamente se realiza una transferencia de conocimientos.

Este grupo de imágenes es con el que verdaderamente entrenamos el clasificador. Dentro del ejemplo de transfer learning que nos proporciona Microsoft encontramos el código necesario para obtener este dataset de imágenes. Utilizamos este dataset por la facilidad que nos aporta a la hora de obtener las imágenes, también podríamos haber empleado ImageNet y realizar nosotros mismo la creación de este dataset.

Accediendo de forma remota a nuestra máquina virtual, proceso que explicaremos más adelante, podemos ver la dimensión de este dataset, en la

Fig. 6.7 podemos ver enmarcado en rojo la última imagen, que corresponde a la imagen 8189.

Fig. 6. 7 Dataset de imágenes de entrenamiento previo

6.2.2. Entranamiento y evaluación

Para realizar el entrenamiento de nuestro clasificador y su posterior testeo debemos de conectarnos a nuestra máquina virtual, para ello utilizaremos un programa llamado PuTTY, el cual nos permite realizar una conexión SSH con nuestra máquina virtual.

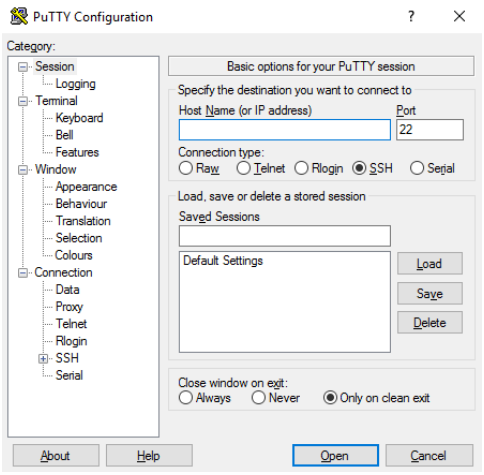

Fig. 6. 8 Pantalla principal PuTTY

Una vez conectados a nuestra maquina lo primero que debemos hacer es actualizar el sistema para evitar cualquier tipo de incoherencia o problema. Seguido de esta actualización debemos de transferir a nuestra maquina los scripts e imágenes para realizar los entrenamientos y evaluaciones.

Para realizar esta transferencia utilizaremos un repositorio de *GitHub*. El por qué utilizamos GitHub es simple, es debido a que ya lo hemos empleado muchas veces durante el grado en diversas asignaturas además evitamos tener que buscar otro sistema que pueda resultar más complicado.

Primero de todo instalaremos GitHub en la máquina virtual con el siguiente comando: *sudo apt-get install git*. Una vez instalado realizaremos un *git clone* de

nuestro repositorio, el comando es el siguiente: *git clone [nombre repositorio]*. Como podemos ver en la **Fig. 6.9** ya tenemos la carpeta con los archivos necesarios.

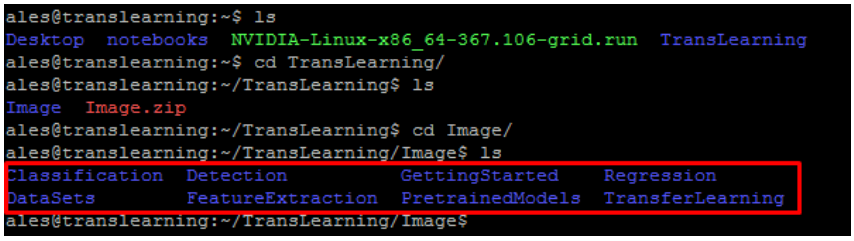

Fig. 6. 9 Directorio dentro de VM para Transfer Learning

Ahora necesitamos descargar el dataset de imágenes con el que realmente entrenaremos nuestro clasificador, para ello ejecutaremos el siguiente script en Python: install_data_and_model.py[1].

Una vez tenemos todos los recursos necesarios para comenzar con el entrenamiento ejecutaremos el script llamado TransferLearning.py[1], que entrenará nuestro clasificador con las más de 8000 imágenes que tenemos. Realmente el entrenamiento se realiza con 5000 imágenes y se evalua con el resto de imágenes.

Este entrenamiento es de una 20 *epochs*, esto quiere decir que los pesos se recalcularán unas 20 veces, este proceso es similar a los *stage* en Haar-cascade, mientras más *epochs* mayor es la precisión. En la **Fig. 6.10** podemos ver el inicio de este entrenamiento.

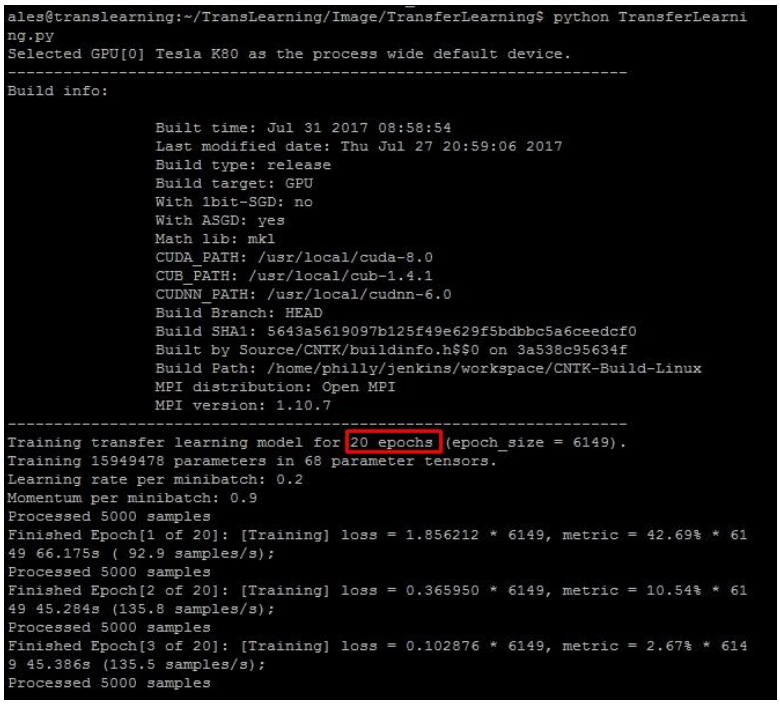

Fig. 6. 10 Entrenamiento del clasificador con 8000 imágenes

1. Anexo V: Transfer learning scripts

Gracias a este entrenamiento y evaluación obtenemos un fichero denominado *Output* que contendrá los conocimientos adquiridos de este entrenamiento, a partir de este fichero se realizaran los entrenamientos con las otras imágenes.

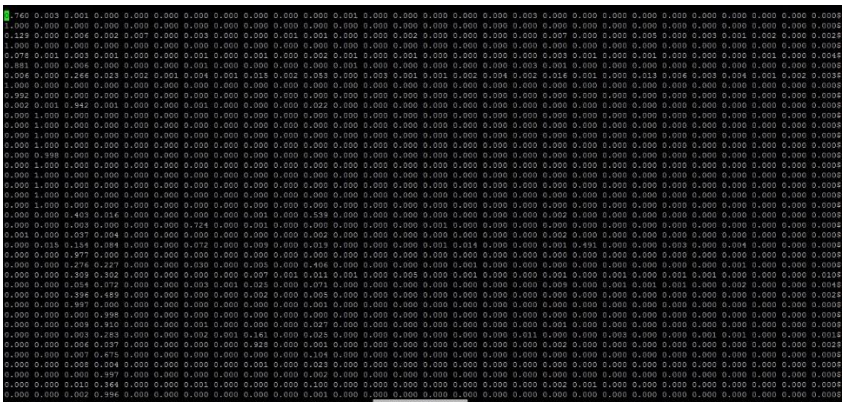

Fig. 6. 11 Fichero *Output*

A partir de este primer entrenamiento realizaremos los entrenamientos respecto nuestros propios objetos a detectar. Para realizar esto utilizaremos una versión extendida del script anterior (TransferLearning.py) que se llama TransferLearning_Extended.py[1]. Con este segundo script no sólo entrenaremos nuestro clasificador sino también comprobaremos su eficacia.

Primero hemos realizado las pruebas sobre el ejemplo que nos ofrece Microsoft el cual es la detección de lobos y ovejas, en la **Fig. 6.12** podemos ver los resultados.

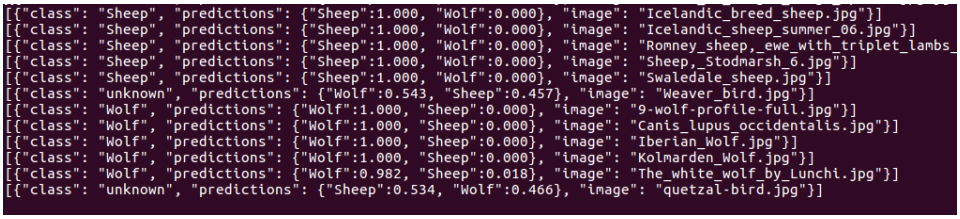

Fig. 6. 12 Evaluación del clasificador para detectar lobos y ovejas

Seguidamente realizamos la prueba con las imágenes de relojes y de señales de aterrizaje para helicópteros, estos resultados podemos verlos en la **Fig. 6.13**.

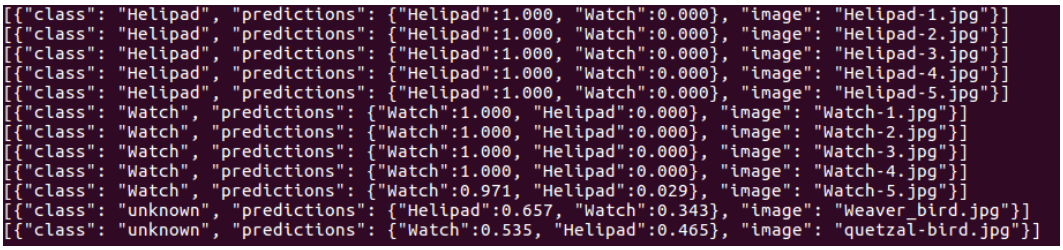

Fig. 6. 13 Evaluación del clasificador para detectar relojes y señales de helipuerto (Helipad)

Si nos fijamos en las **Fig. 6.12** y **Fig. 6.13** podemos ver que hay dos filas en cada imagen donde la clase predicha es desconocida. Esto es debido a que ha tratado

1. Anexo V: Transfer learning scripts

de clasificar las imágenes de los pájaros que habíamos puesto como prueba de error anteriormente. En estos casos el clasificador no termina de decidir de qué tipo de imagen se trata. A diferencia de los otros casos donde clasifica con una eficacia del 100%, excepto en uno que es del 97.1%.

En el caso de los pájaros esta eficacia cae estrepitosamente. Esto da lugar a una predicción del 65.7% que sea una señal de helipuerto y un 34.3% de que sea un reloj, esto en un caso. En el otro caso la predicción es más indecisa. Dando como resultado un 53.5% de que sea un reloj y un 46.5% de que sea una señal de helipuerto. En las conclusiones ya analizaremos con más detalle estos resultados.

[{"class": "Helipad", "predictions": {"Helipad":1.000, "Watch":0.000, "Sheep":0.000}, "image": "Helipad-1.jpg"}]
[{"class": "Helipad", "predictions": {"Helipad":1.000, "Watch":0.000, "Sheep":0.000}, "image": "Helipad-2.jpg"}]
[{"class": "Helipad", "predictions": {"Helipad":1.000, "Watch":0.000, "Wolf":0.000}, "image": "Helipad-3.jpg"}]
[{"class": "Helipad", "predictions": {"Helipad":1.000, "Watch":0.000, "Sheep":0.000}, "image": "Helipad-4.jpg"}]
[{"class": "Helipad", "predictions": {"Helipad":1.000, "Watch":0.000, "Sheep":0.000}, "image": "Helipad-5.jpg"}]
[{"class": "Sheep", "predictions": {"Sheep":1.000, "Helipad":0.000, "Wolf":0.000}, "image": "Icelandic_breed_sheep.jpg"}]
[{"class": "Sheep", "predictions": {"Sheep":1.000, "Wolf":0.000, "Watch":0.000}, "image": "Icelandic_sheep_summer_06.jpg"}]
[{"class": "Sheep", "predictions": {"Sheep":1.000, "Helipad":0.000, "Wolf":0.000}, "image": "Romney_sheep,_ewe_with_triplet_
[{"class": "Sheep", "predictions": {"Sheep":1.000, "Wolf":0.000, "Helipad":0.000}, "image": "Sheep,_Stodmarsh_6.jpg"}]
[{"class": "Sheep", "predictions": {"Sheep":1.000, "Watch":0.000, "Helipad":0.000}, "image": "Swaledale_sheep.jpg"}]
[{"class": "Watch", "predictions": {"Watch":1.000, "Helipad":0.000, "Sheep":0.000}, "image": "Watch-1.jpg"}]
[{"class": "Watch", "predictions": {"Watch":0.997, "Helipad":0.003, "Sheep":0.000}, "image": "Watch-2.jpg"}]
[{"class": "Watch", "predictions": {"Watch":1.000, "Helipad":0.000, "Sheep":0.000}, "image": "Watch-3.jpg"}]
[{"class": "Watch", "predictions": {"Watch":1.000, "Helipad":0.000, "Sheep":0.000}, "image": "Watch-4.jpg"}]
[{"class": "Watch", "predictions": {"Watch":0.739, "Helipad":0.261, "Sheep":0.000}, "image": "Watch-5.jpg"}]
[{"class": "Wolf", "predictions": {"Wolf":1.000, "Helipad":0.000, "Sheep":0.000}, "image": "9-wolf-profile-full.jpg"}]
[{"class": "Wolf", "predictions": {"Wolf":1.000, "Helipad":0.000, "Sheep":0.000}, "image": "Canis_lupus_occidentalis.jpg"}]
[{"class": "Wolf", "predictions": {"Wolf":1.000, "Helipad":0.000, "Sheep":0.000}, "image": "Iberian_Wolf.jpg"}]
[{"class": "Wolf", "predictions": {"Wolf":1.000, "Helipad":0.000, "Sheep":0.000}, "image": "Kolmarden_Wolf.jpg"}]
[{"class": "Wolf", "predictions": {"Wolf":1.000, "Sheep":0.000, "Helipad":0.000}, "image": "The_white_wolf_by_Lunchi.jpg"}]

Fig. 6. 14 Evaluación del clasificador con diferentes grupos de imágenes

En este último caso que se refleja en la **Fig. 6.14** podemos ver como el clasificador es capaz de clasificar sin problema cada imagen. Con una eficacia del 100%.

Únicamente podemos destacar dos casos que podemos ver enmarcados en rojo, donde confunde ligeramente imágenes de relojes con señales de helipuerto.

En un caso con 73.9% de que sea un reloj y un 26.1% de que sea una señal, y en otro con un 99.7% de que sea un reloj y un 0.3% de que sea una señal de helipuerto. En ambos casos la imagen es de un reloj de manera que la clasificación es correcta.

El caso donde la clasificación es de un 73.9% corresponde a la misma imagen donde en la **Fig. 6.13** la clasificación tuvo un acierto del 97.1%.

Esto puede ser debido a la imagen o al haber más clases (lobos y ovejas) que únicamente relojes y señales de helipuerto. Esto puede hacer que al clasificador le cuesta más decidir.

Viendo la última columna donde podemos ver el nombre de la imagen confirmamos que las imágenes son de relojes. Por tanto el clasificador acertó en su predicción.

6.3. Conclusiones

De esta segunda fase experimental podemos extraer varias conclusiones.

La primera de ella es que ofrece una gran precisión a la hora de clasificar. Los resultados lo demuestran con una precisión del 100% excepto algún caso excepcional. También podemos ver, que si la predicción que nos ofrece no obtiene unos resultados semejantes al 90% es muy probable que esta imagen no pertenezca a ninguna de las clases que tenemos como baremo. Esto debido a su alta precisión.

Pero por contra partida nos encontramos que necesitamos unos recursos computacionales bastante grandes. Esto nos ha creado diversas difcultades.

Una de ellas es la dificultad que ha conllevado preparar todo el entorno. Además de algunos problemas añadidos con Azure, debido a la activación de la cuenta. Esto ha hecho imposible extraer esta clasificación a una detección sobre una imagen, enmarcando el objeto detectado y reflejando de forma más evidente la detección.

También hemos probado a combinar transfer learning con otras tecnologías para realizar esta detección más concreta. Esta idea se comentará más adelante en las conclusiones del trabajo.

CONCLUSIONES

Como conclusión hay diferentes temas a abordar. Entre ellos el haber conseguido cumplir los objetivos, algo que no ha sido del todo posible.

Primero hay que hablar que en este proyecto hemos cubierto de forma teórica los dos métodos de detección de objetos. La dificultad que con lleva entender el funcionamiento de cada método es muy grande, sobre todo debido a la complejidad del funcionamiento.

Por un lado hemos chocado con la dificultad de entender cómo funciona a nivel teórico Haar-Cascade, el cual emplea filtros para tratar las imágenes y un clasificador posterior que ha de ser un AdaBoost debido a su velocidad. También hemos comprobado a nivel teórico como aplican los pesos sobre las características para así poder diferenciar y clasificar con mayor eficacia. Luego hemos podido ver que todos estos procesos se realizan de forma iterativa en forma de cascada.

A consecuencia de todo esto hemos puesto en práctica esta teoría utilizando OpenCV una librería que nunca habíamos utilizado antes. OpenCV nos ha permitido tratar las imágenes dentro de las diversas pruebas realizadas con relojes por un lado y señales de aterrizaje de helicópteros por otro, y nos hemos encontrado con dos puntos importantes.

El primero la capacidad de procesado para la detección de objetos empleando un número ínfimo de imágenes positivas resulta muy elevado, haciendo poco eficaz la detección sino se tiene la suficiente. El segundo punto es que utilizar métodos alternativos como merge. Donde se realiza una unión de dos archivos con los vectores de las imágenes positivas tampoco aumenta en gran medida ni precisión ni la sensibilidad. En algunos casos como en los relojes distorsionó las pruebas y dio peores resultados.

Un problema que hemos tenido en cada prueba es el tiempo necesario para entrenar el clasificador, que podían durar desde 6 horas a 10 dependiendo el número de stages que hagamos, esto en haar-cacade. En el caso de transfer learning el proceso se realizaba en cuestión una hora aproximadamente.

En la segunda fase experimental los resultados fueron muy positivos. Pero primero era necesario entender que estábamos haciendo y que está sucediendo, por ello estudiamos de forma teórica el proceso de transfer learning descubriendo un nuevo framework, Cognitive Toolkit.

También hemos visto cómo funciona el clasificador DCNN. Para explicar este clasificador de forma teórica era más sencillo conocer el funcionamiento de un clasificador CNN, donde la única diferencia que tienen son los niveles de profundidad. Estos niveles son debidos a la infraestructura y red cognitiva que ofrece Microsoft con Cognitive Toolkit.

Estudiando CNN hemos visto la complejidad que suponen las redes neuronales, las diferentes funciones que emplean y que requieren una infraestructura interna muy compleja para obtener unos resultados positivos.

Después de haber visto de forma teórica el funcionamiento de los clasificadores CNN. Hemos realizado las purebas a nivel experimental con transfer learning. Para ello hemos necesitado de un servicio de virtualización de Azure.

Hemos descubierto unos nuevos servicios que no simplemente se trata de una máquina virtual como hemos utilizado normalmente durante el grado en diferentes asignaturas, sino servicios más completos que nos ofrecen una gran potencia de procesado con procesadores dedicados llamados GPU.

Estos nuevos servicios nos han permitido obtener muy buenos resultados, pero nos ha faltado llegar un paso más allá pudiendo obtener resultados visibles como en la primera fase experimental, en la cual recuadrábamos el objeto detectado.

Una idea planteada durante la realización del proyecto, pero que se descartó por la dificultad que suponía, era emplear *Fast R-CNN* para detectar los objetos.

La idea consistía en adaptar el modelo obtenido con transfer learning al algoritmo Fast R-CNN. Después de un par de semanas de pruebas comprobamos que realizar esta adaptación era muy complicada y no conseguíamos avanzar. Esto provocó el abandono de esta idea debido a su dificultad.

Otra idea que surge a raíz de la segunda fase experimental, es utilizar transfer learning para aumentar el número de imágenes positivas. Esto lo haríamos aplicándolo sobre un directorio de imágenes y utilizando su gran precisión y sensibilidad a la hora de clasificar.

Como idea final podemos decir que ambas tecnologías ofrecen un gran futuro, pero que transfer learning va un paso por adelante según mi opinión, ya que los resultados numéricos obtenidos son muy buenos. Sigue existiendo siempre algún problema como saber si es posible con este método capturar un flujo de video y analizarlo al momento.

De cara al futuro son algunas cosas que pueden cambiar, la tecnología avanza muy rápido. Las tarjetas gráficas creadas por empresas como nvidia tienen una capacidad asombrosa permitiendo realizar cálculos que suponen horas en minutos. También si se puede combinar con otros algoritmos como intentamos con *Fast R-CNN* podemos obtener grandes resultados.

BIBLIOGRAFÍA

[1] Object Recognition: History and overview [En línea] [Fecha de consulta: Febrero de 2017]. Enlace:
http://www.cs.unc.edu/~lazebnik/spring10/lec16_recognition_intro.pdf

[2] The Evolution of Object Recognition in Embedded Computer Vision [En línea] [Fecha de consulta: Febrero de 2017]. Enlace:
https://www.design-reuse.com/articles/37837/object-recognition-in-embedded-computer-vision.html

[3] Image recognition and object detection [En línea] [Fecha de consulta: Febrero de 2017]. Enlace:
https://www.learnopencv.com/image-recognition-and-object-detection-part1/

[4] Generic object detection with deformable part-based models [En línea] [Fecha de consulta: Septiembre de 2017]. Enlace:
http://slideplayer.com/slide/6030352/

[5] Face recognition dataset [En línea] [Fecha de consulta: Febrero de 2017]. Enlace: http://chrisdecoro.com/eigenfaces/

[6] Robot vision vs Computer vision: What's the difference? [En línea] [Fecha de consulta: Marzo de 2017]. Enlace:
https://blog.robotiq.com/robot-vision-vs-computer-vision-whats-the-difference

[7] Computer vision evolution and promise [En línea] [Fecha de consulta: Marzo de 2017]. Enlace:
https://cds.cern.ch/record/400313/files/p21.pdf

[8] Computer vision vs machine vision [En línea] [Fecha de consulta: Marzo de 2017]. Enlace:
https://www.visiononline.org/vision-resources-details.cfm/vision-resources/Computer-Vision-vs-Machine-Vision/content_id/4585

[9] Supervised learning [En línea] [Fecha de consulta: Marzo de 2017]. Enlace:
https://en.wikipedia.org/wiki/Supervised_learning

[10] Unsupervised learning [En línea] [Fecha de consulta: Marzo de 2017]. Enlace:
https://en.wikipedia.org/wiki/Unsupervised_learning

[11] Supervised and unsupervised machine learnings algorithms [En línea] [Fecha de consulta: Marzo de 2017]. Enlace:
https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[12] Clasificación supervisada y no supervisada [En línea] [Fecha de consulta: Marzo de 2017]. Enlace:

http://www.cs.us.es/~fsancho/?e=77

[13] What's the difference between a supervised and unsupervised image classification? [En línea] [Fecha de consulta: Marzo de 2017]. Enlace: https://articles.extension.org/pages/40214/whats-the-difference-between-a-supervised-and-unsupervised-image-classification

[14] Semi-supervised clustering [En línea] [Fecha de consulta: Marzo de 2017]. Enlace: http://www.cs.upc.edu/~bejar/amlt/material/06a-SemisupervisedClustering.pdf

[15] Semi-supervised learning [En línea] [Fecha de consulta: Marzo de 2017]. Enlace: https://en.wikipedia.org/wiki/Semi-supervised_learning

[16] stackoverflow: detect tan one object on a picture [En línea] [Fecha de consulta: Marzo de 2017]. Enlace: https://stackoverflow.com/questions/41160721/detect-more-than-1-object-on-picture

[17] OpenCv: Librería de visión por computador [En línea] [Fecha de consulta: Marzo de 2017]. Enlace: https://osl.ull.es/software-libre/opencv-libreria-vision-computador/

[18] OpenCv [En línea] [Fecha de consulta: Marzo de 2017]. Enlace: https://es.wikipedia.org/wiki/OpenCV

[19] Face detection using Haar cascade [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

[20] Object category detection: sliding windows [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://courses.engr.illinois.edu/cs543/sp2011/lectures/Lecture%2019%20-%20Sliding%20Window%20Detection%20-%20Vision_Spring2011.pdf

[21] Training Haar-Cascade [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://singhgaganpreet.wordpress.com/2012/10/14/training-haar-cascade/

[22] What is "Histogram of Oriented Gradients" and how does it work? [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://www.quora.com/What-is-Histogram-of-Oriented-Gradients-and-how-does-it-work

[23] Is Haar Cascade the only available technique for image recognition in OpenCV? [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://stackoverflow.com/questions/7601413/is-haar-cascade-the-only-available-technique-for-image-recognition-in-opencv

[24] Detector de caras basado en filtros de Haar + Adaboost [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://es.coursera.org/learn/deteccion-objetos/lecture/eczp1/l5-1-detector-de-caras-basado-en-filtros-de-haar-adaboost

[25] Detección de objetos [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://es.coursera.org/learn/deteccion-objetos

[26] Imagen integral [En línea] [Fecha de consulta: Abril de 2017]. Enlace: https://es.coursera.org/learn/deteccion-objetos/lecture/CROQ4/l5-3-imagen-integral

[27] What is Adaboost? [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://www.quora.com/What-is-AdaBoost

[28] Adaboost [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://es.coursera.org/learn/deteccion-objetos/lecture/KMPxn/l5-4-adaboost

[29] Computer Vision: What is the difference between HOG and SIFT feature descriptor? [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://www.quora.com/Computer-Vision-What-is-the-difference-between-HOG-and-SIFT-feature-descriptor

[30] Feature detection description [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://docs.opencv.org/trunk/db/d27/tutorial_py_table_of_contents_feature2d.html

[31] Why do we use keypoint descriptors? [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://dsp.stackexchange.com/questions/10423/why-do-we-use-keypoint-descriptors

[32] Introduction to SURF (Speeded-Up Robust Features) [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

[33] Introduction to SIFT (Scale-Invariant Feature Transform) [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html?highlight=sift

[34] What are the best pattern matching algorithms in OpenCV? Is there an algorithm where I can train on one model instead of a data set? [En línea] [Fecha de consulta: Mayo de 2017]. Enlace: https://www.quora.com/What-are-the-best-pattern-matching-algorithms-in-OpenCV-Is-there-an-algorithm-where-I-can-train-on-one-model-instead-of-a-data-set

[35] Ubuntu 16.04: How to install OpenCV [En línea] [Fecha de consulta: Junio de 2017]. Enlace:
https://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/

[36] Install OpenCV on Ubuntu or Debian [En línea] [Fecha de consulta: Junio de 2017]. Enlace:
http://milq.github.io/install-opencv-ubuntu-debian/

[37] How to detect object and track with OpenCV [En línea] [Fecha de consulta: Junio de 2017]. Enlace:
https://www.intorobotics.com/how-to-detect-and-track-object-with-opencv/

[38] Creating your own Haar Cascade OpenCV Python Tutorial [En línea] [Fecha de consulta: Junio de 2017]. Enlace:
https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/

[39] ImageNet [En línea] [Fecha de consulta: Junio de 2017]. Enlace:
http://www.image-net.org/

[40] Train your own OpenCV Haar classifier [En línea] [Fecha de consulta: Junio de 2017]. Enlace:
http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html

[41] Cascade classifier training [En línea] [Fecha de consulta: Julio de 2017]. Enlace:
https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html

[42] Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features) [En línea] [Fecha de consulta: Julio de 2017]. Enlace:
http://note.sonots.com/SciSoftware/haartraining.html

[43] Creating a Cascade of Haar-Like Classifiers: Step by Step [En línea] [Fecha de consulta: Julio de 2017]. Enlace:
https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Creating_a_Cascade_of_Haar-Like_Classifiers_Step_by_Step.pdf

[44] Cascade classifier training [En línea] [Fecha de consulta: Julio de 2017]. Enlace:
https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html

[45] How do I run opencv_createsamples with multiple positive images? [En línea] [Fecha de consulta: Julio de 2017]. Enlace:
http://answers.opencv.org/question/29765/how-do-i-run-opencv_createsamples-with-multiple-positive-images/

[46] Mergevec [En línea] [Fecha de consulta: Julio de 2017]. Enlace:
https://github.com/wulfebw/mergevec

[47] CNTK 301: Image Recognition with Deep Transfer Learning [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://github.com/Microsoft/CNTK/blob/v2.0/Tutorials/CNTK_301_Image_Recognition_with_Deep_Transfer_Learning.ipynb

[48] Deep Learning with Microsoft CNTK [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://www.packtpub.com/books/content/deep-learning-microsoft-cntk

[49] Computational Neural Toolkit [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://www.cntk.ai/Features/Index.html

[50] What's Deep Learning? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://machinelearningmastery.com/what-is-deep-learning/

[51] ¿Qué es la computación acelerada por GPU? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
http://la.nvidia.com/object/what-is-gpu-computing-la.html

[52] En que se diferencia la GPU de CPU [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
http://es.gizmodo.com/en-que-se-diferencia-la-gpu-de-la-cpu-explicado-en-cin-1780816080

[53] What is the difference between a neural network and a deep neural network? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network

[54] Convolutional Neural Network [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://en.wikipedia.org/wiki/Convolutional_neural_network

[55] CNTK FAQ [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://docs.microsoft.com/en-us/cognitive-toolkit/CNTK-FAQ

[56] What is an intuitive explanation of Convolutional Neural Networks? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://www.quora.com/What-is-an-intuitive-explanation-of-Convolutional-Neural-Networks

[57] A quick introduction to Neural Networks [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

[58] Understanding Convolutional Neural Networks for NLP [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[59] Why do we use ReLU in neural networks and how do we use it? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://stats.stackexchange.com/questions/226923/why-do-we-use-relu-in-neural-networks-and-how-do-we-use-it

[60] Neural Networks [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

[61] Stochastic gradient descent [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[62] Recurrent neural network [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://en.wikipedia.org/wiki/Recurrent_neural_network

[63] Long short-term memory [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://en.wikipedia.org/wiki/Long_short-term_memory

[64] Long short-term memory [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[65] What is the role of the activation function in a neural network? How does this function in a human neural network system? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://www.quora.com/What-is-the-role-of-the-activation-function-in-a-neural-network-How-does-this-function-in-a-human-neural-network-system

[66] A Beginner's Guide to Understanding Convolutional Neural Networks [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[67] A Beginner's Guide to Understanding Convolutional Neural Networks Part 2 [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[68] What is a global average pooling? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://www.quora.com/What-is-global-average-pooling

[69] Max pooling vs Sum pooling? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace: https://stackoverflow.com/questions/37434426/max-pooling-vs-sum-pooling

[70] Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
http://www.di.ens.fr/~josef/publications/oquab14.pdf

[71] Why does deep learning/architectures only use the non-linear activation function in the hidden layers? [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
https://www.quora.com/Why-does-deep-learning-architectures-only-use-the-non-linear-activation-function-in-the-hidden-layers

[72] Transfer learning – Machine learning's next frontier [En línea] [Fecha de consulta: Agosto de 2017]. Enlace:
http://ruder.io/transfer-learning/

[73] Build your own image classifier using Transfer Learning [En línea] [Fecha de consulta: Septiembre de 2017]. Enlace:
https://docs.microsoft.com/en-us/cognitive-toolkit/build-your-own-image-classifier-using-transfer-learning#using-a-different-base-model

[74] Crash Course On Multi-Layer Perceptron Neural Networks [En línea] [Fecha de consulta: Septiembre de 2017]. Enlace:
https://machinelearningmastery.com/neural-networks-crash-course/

[75] Microsoft Azure [En línea] [Fecha de consulta: Septiembre de 2017]. Enlace:
https://azure.microsoft.com/es-es/?&WT.srch=1&wt.mc_id=AID631176_SEM_QwnrSFdW&gclid=EAIaIQobChMIxOrSh6v61gIVCZ4bCh1snwaGEAAYASAAEgKhyfD_BwE

[76] Deep learning virtual machine [En línea] [Fecha de consulta: Septiembre de 2017]. Enlace:
https://azuremarketplace.microsoft.com/en-us/marketplace/apps/microsoft-ads.dsvm-deep-learning?tab=PlansAndPrice

[77] Aprovisionamiento de Data Science Virtual Machine de Windows en Azure [En línea] [Fecha de consulta: Septiembre de 2017]. Enlace:
https://docs.microsoft.com/es-es/azure/machine-learning/data-science-virtual-machine/provision-vm

[78] Portal Azure [En línea] [Fecha de consulta: Septiembre de 2017]. Enlace:
https://portal.azure.com/

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

TÍTULO DEL TFG: Big data y Deep Learning para la detección de objetos dentro de imágenes

TITULACIÓN: Grado en Ingeniería Telemática

AUTOR: Alessandro Iván Fava Fernández

DIRECTORA: Esther Salamí San Juan

FECHA: 31 de Octubre del 2017

ANEXO I. MULTILAYER PERCEPTRON (MPL)

En este anexo explicaremos que una red neuronal *multilayer perceptron.* Explicaremos concretamente este tipo de red neuronal, ya que son las que se emplean normalmente y también porque es este tipo el que empleamos en el proyecto.

Esta red neuronal a diferencia de una *single perceptron* (red con un único nodo simple) es capaz de aprender utilizando funciones no lineales. La estructura es más compleja que una *single perceptron*, ya que se parte de un único nodo (neurona) por cada entrada y se extiende hacia una segunda capa y finalmente hasta una tercera de salida.

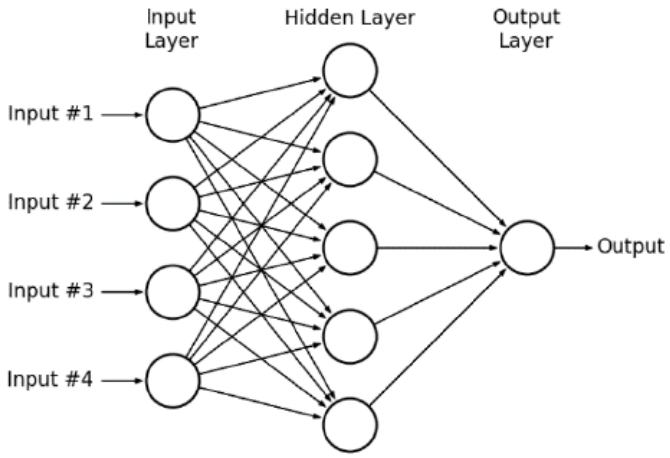

Fig. Anexo I. 1 Representación de una MPL

En este tipo de redes a veces también se emplean *Bias* que son nodos con valores prefijados, que adquieren dos posiciones 1 o 0, de esta forma se asegura un valor de entrada siempre.

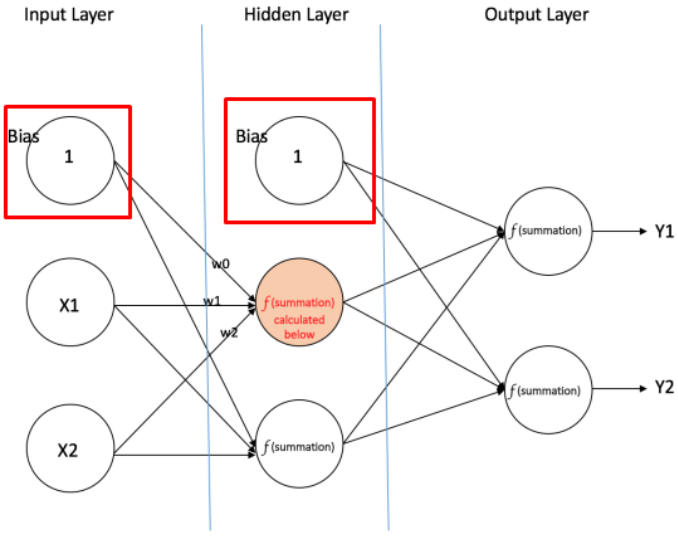

Fig. Anexo I. 2 Representación de una MPL con Bias

Capa de entrada (input layer)

La primera capa también denominada la capa visible se encarga de coger los valores de entrada procedentes del dataset. Normalmente esta capa consta con un nodo de entrada por cada valor de entrada o por cada columna del dataset.

Esta capa como hemos mencionado antes de encarga de coger los valores de entrada y pasarlos a la capa siguiente, la capa oculta (*hidden layer*).

Capa oculta (Hidden layer)

Esta segunda capa denominada capa oculta (*hidden layer*) debido a que no está directamente expuesta los valores de entrada. Esta capa establece relaciones entre las diferentes entradas que pueden ser tanto calores del dataset como en nuestro caso características de una imagen.

Empleando una gran potencia de procesado y eficientes librerías se puede construir una gran capa, esto sucede en los casos de *Deep learning* como CNTK.

Capa de salida (Output layer)

La última capa conocida como capa de salida (*output layer*), en esta capa obtenemos los valores o vector de valores correspondiente con el problema de entrada planteado a la red.

El tipo de salida depende básicamente del tipo de función de activación que apliquemos en esta capa. Debido que aplicamos funciones de activación las cuales retornan respuestas no lineares podemos comprobar que este tipo de red es perfecta para tratar con ellos.

- Si se trata de un problema de regresión con una única neurona de salida es muy probable que no sea necesaria ninguna función de activación.

- Si se trata de un problema de clasificación con una única neurona de salida se suele usar la función *sigmoid* que nos da como salida un valor entre 0 y 1, que es la probabilidad de predicción hacia una clase. Aplicando un valor de umbral podemos decidir en que clase clasificar la salida.

- En el caso de una clasificación con múltiples clases, la función de activación que se suele emplear es la *softmax*, la cual se acostumbra a utilizar para obtener un valor de salida de la predicción de acuerdo con cada clase y empleando un umbral para las salidas con mayor probabilidad.

ANEXO II. INSTALACIÓN OPENCV

La instalación de OpenCV no es algo complejo pero requiere de diferentes librerías para poder tratar los diferentes formatos de las imágenes.

Primero de todo debemos de actualizar nuestro sistema para así cualquier problema con las dependencias o referencias.

$sudo apt-get update
$sudo apt-get upgrade

Una vez actualizado procedemos a instalar las herramientas de desarrollo.

$sudo apt-get install build-essential cmake pkg-config

El agregar *cmake pkg-config* en el comando sirve para configurar automáticamente la nuestra *Build* de OpenCV.

Como hemos comentado anteriormente OpenCv trata con imágenes de diferentes formatos (JPEG, PNG, TIFF, etc.) esto hace que sean necesarias ciertas librerías que facilitan el proceso de descarga y decodificación de las imágenes.

$sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev

Ahora que podemos procesar las imágenes sin problema, la cuestión es que sucede con el video. La respuesta es sencilla necesitaremos de otras librerías que nos permiten procesar los diferentes flujos de video así como acceder a los diferentes fotogramas de la cámara.

$sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$sudo apt-get install libxvidcore-dev libx264-dev

La instalación por defecto de OpenCV incorpora un conjunto de herramientas GUI (*Graphical User Interface*) muy limitadas por lo que es necesario añadir otra librería que amplíe el conjunto de herramientas, esta librería se llama GTK.

$sudo apt-get install libgtk-3-dev

Aun asi necesitamos instalar más librerias que optimizan varias funcionalidades de OpenCV, como las operaciones con matrices.

$sudo apt-get install libatlas-base-dev gfortran

El último punto es comprovar nuestra versión de *python*, que si tenemos actualizado nuestro sistema operativo no habra ningún problema, ya que ubuntu incorpora este lenguaje así como sus librerías.

$python -v

Una vez instaladas todas las dependencias y diferentes librerías necesarias, ya podemos proceder a instalar OpenCV. Obtendremos la *release* de un repositorio de GitHub, la cual deberemos descomprimir.

$wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.1.0.zip
$unzip opencv.zip

Una vez descargada la *release* debemos de descargar la contribución antes de poder instal·lar OpenCV.

$Wget -O opencv_contrib.zip https://github.com/Itseez/opencv_contrib/archive/3.1.0.zip
$unzip opencv_contrib.zip

Finalmente ya tenemos instalado OpenCV y podemos emplear las diferentes funcionalidades dentro de los script en python.

Hay que tener en cuenta que pude que algunas librerías que han sido instaladas no sean necesarias, pero debido a que no sabíamos el alcance de este proyecto es preferible instalar todas.

ANEXO III. HAAR-CASCADE SCRIPTS

En este anexo se encuentran todos los scripts utilizados tanto para entrenar como para testear nuestro clasificador.

```
import urllib.request
import cv2
import numpy as np
import os

def store_raw_images():
    neg_images_link = 'http://image-net.org/api/text/imagenet.synset.geturls?wnid=n07942152'
    neg_image_urls = urllib.request.urlopen(neg_images_link).read().decode()
    pic_num = 1

    if not os.path.exists('neg'):
        os.makedirs('neg')

    for i in neg_image_urls.split('\n'):
        try:
            print(i)
            urllib.request.urlretrieve(i, "neg/"+str(pic_num)+".jpg")
            img = cv2.imread("neg/"+str(pic_num)+".jpg",cv2.IMREAD_GRAYSCALE)
            # should be larger than samples / pos pic (so we can place our image on it)
            resized_image = cv2.resize(img, (100, 100))
            cv2.imwrite("neg/"+str(pic_num)+".jpg",resized_image)
            pic_num += 1

        except Exception as e:
            print(str(e))


store_raw_images()
```

Synset URL

Fig. Anexo III. 1 Script en python para descargar y reajustar las imágenes del synset

```
import urllib.request
import cv2
import numpy as np
import os

def find_uglies():
    match = False
    for file_type in ['neg']:
        for img in os.listdir(file_type):
            for ugly in os.listdir('uglies'):
                try:
                    current_image_path = str(file_type)+'/'+str(img)
                    ugly = cv2.imread('uglies/'+str(ugly))
                    question = cv2.imread(current_image_path)
                    if ugly.shape == question.shape and not(np.bitwise_xor(ugly,question).any()):
                        print('That is one ugly pic! Deleting!')
                        print(current_image_path)
                        os.remove(current_image_path)
                except Exception as e:
                    print(str(e))


find_uglies()
```

Fig. Anexo III. 2 Script para encontrar y eliminar imágenes *feas*

```
import urllib.request
import cv2
import numpy as np
import os

def create_pos_n_neg():
    for file_type in ['neg']:

        for img in os.listdir(file_type):

            if file_type == 'pos':
                line = file_type+'/'+img+' 1 0 0 50 50\n'
                with open('info.dat','a') as f:
                    f.write(line)
            elif file_type == 'neg':
                line = file_type+'/'+img+'\n'
                with open('bg.txt','a') as f:
                    f.write(line)



create_pos_n_neg()
```

Fig. Anexo III. 3 Script para crear archivo *bg.txt*

```
import cv2
import numpy as np
import os


#this is the cascade we just made. Call what you want
def store_results_images():
    pic_num = 1

    haar_cascade = cv2.CascadeClassifier('cascade15stage.xml')

    for file_type in ['samples']:
        for img in os.listdir(file_type):

            cap = cv2.imread(str(file_type)+'/'+str(img),1)

            gray = cv2.cvtColor(cap, cv2.COLOR_BGR2GRAY)

            object = haar_cascade.detectMultiScale(gray, 1.3, 5)

            for (x,y,w,h) in object:
                cv2.rectangle(cap,(x,y),(x+w,y+h),(255,255,0),2)

            cv2.imwrite("results/"+str(pic_num)+".jpg", cap)

            pic_num += 1



store_results_images()
```

Fig. Anexo III. 4 Script para usar el clasificador sobre las imágenes de test y guardar los resultados en una carpeta

```
import numpy as np
import cv2


#this is the cascade we just made. Call what you want
object_cascade = cv2.CascadeClassifier('cascade15stage.xml')

cap = cv2.VideoCapture(0)

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # add this
    # image, reject levels level weights.
    objects = object_cascade.detectMultiScale(gray, 30, 30)

    # add this
    for (x,y,w,h) in watches:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)

    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Fig. Anexo III. 5 Script para detección a tiempo real empleando una webcam

ANEXO IV. EVALUACIÓN DEL RENDIMIENTO DE UN CLASIFICADOR

En este anexo explicaremos como se evalúa el rendimiento de un clasificador. Esto también es aplicable a la hora de evaluar el rendimiento del detector.

Después de la clasificación encontramos diferentes resultados.

Tabla. Anexo IV. 1 Resultados de la clasificación

		Resultado clasificación	
		Objeto	No-objeto
Muestra real	Objeto	Positivos reales	Falsos negativos
	No-objeto	Falsos positivos	Negativos reales

A partir de los resultados podemos extraer dos parámetros que definen el comportamiento de los resultados:

- **Precisión.** Este parámetro mide la calidad de respuesta del clasificador.

$$Precisión = \frac{Positivos\ reales}{Positivos\ reales + Falsos\ positivos}$$

Fig. Anexo IV. 1 Expresión para calcular la precisión de un clasificador

- **Sensibilidad (*recall*).** Mediante este parámetro medimos la eficiencia a la hora de clasificar los elementos.

$$Sensibilidad = \frac{Positivos\ reales}{Positivos\ reales + Falsos\ negativos}$$

Fig. Anexo IV. 2 Expresión para calcular la sensibilidad de un clasificador

ANEXO V. TRANSFER LEARNING SCRIPTS

Microsft nos ofrece el código necesario para ejecutar una detección e objetos mediante transfer learning, pero siempre hay que hacer una serie de cambios así como identificar cuáles son los elementos que debemos emplear. A continuación encontraremos el código empleado en la fase de experimental II de este proyecto.

TransferLearning.py

```
from __future__ import print_function
import numpy as np
import cntk as C
import os
from PIL import Image
from cntk.device import try_set_default_device, gpu
from cntk import load_model, placeholder, Constant
from cntk import Trainer, UnitType
from cntk.logging.graph import find_by_name, get_node_outputs
from cntk.io import MinibatchSource, ImageDeserializer, StreamDefs, StreamDef
import cntk.io.transforms as xforms
from cntk.layers import Dense
from cntk.learners import momentum_sgd, learning_rate_schedule, momentum_schedule
from cntk.ops import combine, softmax
from cntk.ops.functions import CloneMethod
from cntk.losses import cross_entropy_with_softmax
from cntk.metrics import classification_error
from cntk.logging import log_number_of_parameters, ProgressPrinter


################################################
################################################
# general settings
make_mode = False
freeze_weights = False
base_folder = os.path.dirname(os.path.abspath(__file__))
tl_model_file = os.path.join(base_folder, "Output", "TransferLearning.model")
output_file = os.path.join(base_folder, "Output", "predOutput.txt")
features_stream_name = 'features'
label_stream_name = 'labels'
new_output_node_name = "prediction"

# Learning parameters
max_epochs = 20
mb_size = 50
lr_per_mb = [0.2]*10 + [0.1]
momentum_per_mb = 0.9
l2_reg_weight = 0.0005

# define base model location and characteristics
_base_model_file = os.path.join(base_folder, "..", "PretrainedModels", "ResNet18_ImageNet_CNTK.model")
_feature_node_name = "features"
_last_hidden_node_name = "z.x"
_image_height = 224
_image_width = 224
_num_channels = 3
```

```
# define data location and characteristics
_data_folder = os.path.join(base_folder, "..", "DataSets", "Flowers")
_train_map_file = os.path.join(_data_folder, "6k_img_map.txt")
_test_map_file = os.path.join(_data_folder, "1k_img_map.txt")
_num_classes = 102
################################################
################################################


# Creates a minibatch source for training or testing
def create_mb_source(map_file, image_width, image_height, num_channels, num_classes, randomize=True):
    transforms = [xforms.scale(width=image_width, height=image_height, channels=num_channels, interpolations='linear')]
    return MinibatchSource(ImageDeserializer(map_file, StreamDefs(
            features =StreamDef(field='image', transforms=transforms),
            labels   =StreamDef(field='label', shape=num_classes))),
            randomize=randomize)


# Creates the network model for transfer learning
def create_model(base_model_file, feature_node_name, last_hidden_node_name, num_classes, input_features, freeze=False):
    # Load the pretrained classification net and find nodes
    base_model   = load_model(base_model_file)
    feature_node = find_by_name(base_model, feature_node_name)
    last_node    = find_by_name(base_model, last_hidden_node_name)

    # Clone the desired layers with fixed weights
    cloned_layers = combine([last_node.owner]).clone(
        CloneMethod.freeze if freeze else CloneMethod.clone,
        {feature_node: placeholder(name='features')})

    # Add new dense layer for class prediction
    feat_norm  = input_features - Constant(114)
    cloned_out = cloned_layers(feat_norm)
    z          = Dense(num_classes, activation=None, name=new_output_node_name) (cloned_out)

    return z
# Trains a transfer learning model
def train_model(base_model_file, feature_node_name, last_hidden_node_name,
                image_width, image_height, num_channels, num_classes, train_map_file,
                num_epochs, max_images=-1, freeze=False):
    epoch_size = sum(1 for line in open(train_map_file))
    if max_images > 0:
        epoch_size = min(epoch_size, max_images)

    # Create the minibatch source and input variables
    minibatch_source = create_mb_source(train_map_file, image_width, image_height, num_channels, num_classes)
    image_input = C.input_variable((num_channels, image_height, image_width))
    label_input = C.input_variable(num_classes)

    # Define mapping from reader streams to network inputs
    input_map = {
        image_input: minibatch_source[features_stream_name],
        label_input: minibatch_source[label_stream_name]
    }

    # Instantiate the transfer learning model and loss function
    tl_model = create_model(base_model_file, feature_node_name, last_hidden_node_name, num_classes, image_input, freeze)
    ce = cross_entropy_with_softmax(tl_model, label_input)
    pe = classification_error(tl_model, label_input)

    # Instantiate the trainer object
    lr_schedule = learning_rate_schedule(lr_per_mb, unit=UnitType.minibatch)
    mm_schedule = momentum_schedule(momentum_per_mb)
    learner = momentum_sgd(tl_model.parameters, lr_schedule, mm_schedule, l2_regularization_weight=l2_reg_weight)
    progress_printer = ProgressPrinter(tag='Training', num_epochs=num_epochs)
    trainer = Trainer(tl_model, (ce, pe), learner, progress_printer)

    # Get minibatches of images and perform model training
    print("Training transfer learning model for {0} epochs (epoch_size = {1}).".format(num_epochs, epoch_size))
    log_number_of_parameters(tl_model)
    for epoch in range(num_epochs):       # loop over epochs
        sample_count = 0
        while sample_count < epoch_size:  # loop over minibatches in the epoch
            data = minibatch_source.next_minibatch(min(mb_size, epoch_size-sample_count), input_map=input_map)
            trainer.train_minibatch(data)                                    # update model with it
            sample_count += trainer.previous_minibatch_sample_count          # count samples processed so far
            if sample_count % (100 * mb_size) == 0:
                print ("Processed {0} samples".format(sample_count))

        trainer.summarize_training_progress()

    return tl_model
```

```
# Evaluates a single image using the provided model
def eval_single_image(loaded_model, image_path, image_width, image_height):
    # load and format image (resize, RGB -> BGR, CHW -> HWC)
    img = Image.open(image_path)
    if image_path.endswith("png"):
        temp = Image.new("RGB", img.size, (255, 255, 255))
        temp.paste(img, img)
        img = temp
    resized = img.resize((image_width, image_height), Image.ANTIALIAS)
    bgr_image = np.asarray(resized, dtype=np.float32)[..., [2, 1, 0]]
    hwc_format = np.ascontiguousarray(np.rollaxis(bgr_image, 2))

    ## Alternatively: if you want to use opencv-python
    # cv_img = cv2.imread(image_path)
    # resized = cv2.resize(cv_img, (image_width, image_height), interpolation=cv2.INTER_NEAREST)
    # bgr_image = np.asarray(resized, dtype=np.float32)
    # hwc_format = np.ascontiguousarray(np.rollaxis(bgr_image, 2))

    # compute model output
    arguments = {loaded_model.arguments[0]: [hwc_format]}
    output = loaded_model.eval(arguments)

    # return softmax probabilities
    sm = softmax(output[0])
    return sm.eval()
# Evaluates an image set using the provided model
def eval_test_images(loaded_model, output_file, test_map_file, image_width, image_height, max_images=-1, column_offset=0):
    num_images = sum(1 for line in open(test_map_file))
    if max_images > 0:
        num_images = min(num_images, max_images)
    print("Evaluating model output node '{0}' for {1} images.".format(new_output_node_name, num_images))

    pred_count = 0
    correct_count = 0
    np.seterr(over='raise')
    with open(output_file, 'wb') as results_file:
        with open(test_map_file, "r") as input_file:
            for line in input_file:
                tokens = line.rstrip().split('\t')
                img_file = tokens[0 + column_offset]
                probs = eval_single_image(loaded_model, img_file, image_width, image_height)

                pred_count += 1
                true_label = int(tokens[1 + column_offset])
                predicted_label = np.argmax(probs)
                if predicted_label == true_label:
                    correct_count += 1

                np.savetxt(results_file, probs[np.newaxis], fmt="%.3f")
                if pred_count % 100 == 0:
                    print("Processed {0} samples ({1} correct)".format(pred_count, (float(correct_count) / pred_count)))
                if pred_count >= num_images:
                    break

    print ("{0} out of {1} predictions were correct {2}.".format(correct_count, pred_count, (float(correct_count) / pred_count)))
if __name__ == '__main__':
    try_set_default_device(gpu(0))
    # check for model and data existence
    if not (os.path.exists(_base_model_file) and os.path.exists(_train_map_file) and os.path.exists(_test_map_file)):
        print("Please run 'python install_data_and_model.py' first to get the required data and model.")
        exit(0)

    # You can use the following to inspect the base model and determine the desired node names
    # node_outputs = get_node_outputs(load_model(_base_model_file))
    # for out in node_outputs: print("{0} {1}".format(out.name, out.shape))

    # Train only if no model exists yet or if make_mode is set to False
    if os.path.exists(tl_model_file) and make_mode:
        print("Loading existing model from %s" % tl_model_file)
        trained_model = load_model(tl_model_file)
    else:
        trained_model = train_model(_base_model_file, _feature_node_name, _last_hidden_node_name,
                                    _image_width, _image_height, _num_channels, _num_classes, _train_map_file,
                                    max_epochs, freeze=freeze_weights)
        trained_model.save(tl_model_file)
        print("Stored trained model at %s" % tl_model_file)

    # Evaluate the test set
    eval_test_images(trained_model, output_file, _test_map_file, _image_width, _image_height)

    print("Done. Wrote output to %s" % output_file)
```

TransferLearning_Extended.py

```
from __future__ import print_function
import numpy as np
import os
from cntk import load_model
from TransferLearning import *


# define base model location and characteristics
base_folder = os.path.dirname(os.path.abspath(__file__))
base_model_file = os.path.join(base_folder, "..", "PretrainedModels", "ResNet18_ImageNet_CNTK.model")
feature_node_name = "features"
last_hidden_node_name = "z.x"
image_height = 224
image_width = 224
num_channels = 3

# define data location and characteristics
train_image_folder = os.path.join(base_folder, "..", "DataSets", "Animals", "Train")
test_image_folder = os.path.join(base_folder, "..", "DataSets", "Animals", "Test")
file_endings = ['.jpg', '.JPG', '.jpeg', '.JPEG', '.png', '.PNG']

def create_map_file_from_folder(root_folder, class_mapping, include_unknown=False):
    map_file_name = os.path.join(root_folder, "map.txt")
    lines = []
    for class_id in range(0, len(class_mapping)):
        folder = os.path.join(root_folder, class_mapping[class_id])
        if os.path.exists(folder):
            for entry in os.listdir(folder):
                filename = os.path.join(folder, entry)
                if os.path.isfile(filename) and os.path.splitext(filename)[1] in file_endings:
                    lines.append("{0}\t{1}\n".format(filename, class_id))

    if include_unknown:
        for entry in os.listdir(root_folder):
            filename = os.path.join(root_folder, entry)
            if os.path.isfile(filename) and os.path.splitext(filename)[1] in file_endings:
                lines.append("{0}\t-1\n".format(filename))

    lines.sort()
    with open(map_file_name , 'w') as map_file:
        for line in lines:
            map_file.write(line)

    return map_file_name
def create_class_mapping_from_folder(root_folder):
    classes = []
    for _, directories, _ in os.walk(root_folder):
        for directory in directories:
            classes.append(directory)
    classes.sort()
    return np.asarray(classes)

def format_output_line(img_name, true_class, probs, class_mapping, top_n=3):
    class_probs = np.column_stack((probs, class_mapping)).tolist()
    class_probs.sort(key=lambda x: float(x[0]), reverse=True)
    top_n = min(top_n, len(class_mapping)) if top_n > 0 else len(class_mapping)
    true_class_name = class_mapping[true_class] if true_class >= 0 else 'unknown'
    line = '[{"class": "%s", "predictions": {' % true_class_name
    for i in range(0, top_n):
        line = '%s"%s":%.3f, ' % (line, class_probs[i][1], float(class_probs[i][0]))
    line = '%s}, "image": "%s"}]\n' % (line[:-2], img_name.replace('\\', '/').rsplit('/', 1)[1])
    return line
```

```
def train_and_eval(_base_model_file, _train_image_folder, _test_image_folder, _results_file, _new_model_file, testing = False):
    # check for model and data existence
    if not (os.path.exists(_base_model_file) and os.path.exists(_train_image_folder) and os.path.exists(_test_image_folder)):
        print("Please run 'python install_data_and_model.py' first to get the required data and model.")
        exit(0)

    # get class mapping and map files from train and test image folder
    class_mapping = create_class_mapping_from_folder(_train_image_folder)
    train_map_file = create_map_file_from_folder(_train_image_folder, class_mapping)
    test_map_file = create_map_file_from_folder(_test_image_folder, class_mapping, include_unknown=True)

    # train
    trained_model = train_model(_base_model_file, feature_node_name, last_hidden_node_name,
                                image_width, image_height, num_channels,
                                len(class_mapping), train_map_file, num_epochs=30, freeze=True)

    if not testing:
        trained_model.save(_new_model_file)
        print("Stored trained model at %s" % tl_model_file)

    # evaluate test images
    with open(_results_file, 'w') as output_file:
        with open(test_map_file, "r") as input_file:
            for line in input_file:
                tokens = line.rstrip().split('\t')
                img_file = tokens[0]
                true_label = int(tokens[1])
                probs = eval_single_image(trained_model, img_file, image_width, image_height)

                formatted_line = format_output_line(img_file, true_label, probs, class_mapping)
                output_file.write(formatted_line)

    print("Done. Wrote output to %s" % _results_file)

if __name__ == '__main__':
    try_set_default_device(gpu(0))

    results_file = os.path.join(base_folder, "Output", "predictions.txt")
    new_model_file = os.path.join(base_folder, "Output", "TransferLearning.model")
    train_and_eval(base_model_file, train_image_folder, test_image_folder, results_file, new_model_file)
```